UNIVERSIDADE DO VALE DO TAQUARI - UNIVATES
CURSO DE ENGENHARIA DE SOFTWARE

UMA PLATAFORMA *WEB* PARA SUPORTE AO DESENVOLVIMENTO DE APLICAÇÕES COM ÊNFASE NA DISPONIBILIZAÇÃO DE UMA *API REST* PARA ACESSO A BASES DE DADOS POSTGRESQL E MONGODB

Lucas Tomasi

Lajeado, novembro de 2019.

Lucas Tomasi

UMA PLATAFORMA *WEB* PARA SUPORTE AO DESENVOLVIMENTO DE APLICAÇÕES COM ÊNFASE NA DISPONIBILIZAÇÃO DE UMA *API REST* PARA ACESSO A BASES DE DADOS POSTGRESQL E MONGODB

Monografia apresentada ao Centro de Ciências Exatas e Tecnológicas da Universidade do Vale do Taquari - UNIVATES, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof. Me. Fabrício Pretto

Lajeado, novembro de 2019.

Lucas Tomasi

UMA PLATAFORMA *WEB* PARA SUPORTE AO DESENVOLVIMENTO DE APLICAÇÕES COM ÊNFASE NA DISPONIBILIZAÇÃO DE UMA *API REST* PARA ACESSO A BASES DE DADOS POSTGRESQL E MONGODB

A Banca examinadora abaixo, aprova a Monografia apresentada na disciplina Trabalho de Conclusão de Curso II, da Universidade do Vale do Taquari - UNIVATES, como exigência para obtenção do grau de Bacharel em Engenharia de Software:

Prof. Me. Fabrício Pretto – Orientador

Prof. Dr. Evandro Franzen

Prof. Me. Willian Valmorbida

Lajeado, novembro de 2019.

RESUMO

A utilização dos bancos de dados relacionais e não relacionais está crescendo no desenvolvimento de software, possibilitando a escolha do tipo de banco que melhor se ajusta à aplicação que está sendo construída. As tecnologias de banco de dados disponíveis no mercado se diferenciam quanto à forma com que armazenam, acessam e manipulam os dados, não mantendo um padrão entre elas, forçando uma adaptação dos profissionais de desenvolvimento. Com base nessa constatação, torna-se importante a existência de ferramentas que auxiliem as equipes de desenvolvimento, tornando transparente as especificidades dos diversos tipos de banco de dados, diminuindo a complexidade e consequentemente facilitando a utilização destes. Com base nesse cenário, o presente trabalho, teve como objetivo desenvolver uma solução em que a manipulação e o acesso aos dados seja padronizado através de uma *API REST*, tanto para bancos de dados relacionais quanto para bancos de dados não relacionais. A solução desenvolvida foi validada por profissionais que atuam na área de desenvolvimento de software, exibindo resultados otimistas e com probabilidade de se tornar um produto.

Palavras-chave: Engenharia de Software. *API REST*. Banco de dados Relacionais. Banco de dados Não Relacional.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API:	Application Programming Interface
BD:	Banco de Dados
CRUD:	Create, Read, Update e Delete
CSV*:*	Comma Separated Values
DDL:	Data Definition Language
DML:	Data Manipulation Language
ER:	Entidade-Relacionamento
HTML:	Hypertext Markup Language
HTTP:	HyperText Transfer Protocol
JSON:	JavaScript Object Notation
MIME:	Multipurpose Internet Mail Extensions
PHP:	PHP Hypertext Preprocessor
REST:	REpresentational State Transfer

SGBD:	Sistema de Gerenciamento de Banco de Dados
SOAP:	Simple Object Access Protocol
SQL:	Structured Query Language
TI:	Tecnologia da Informação
URI:	Uniform Resource Identifier
XML:	eXtensible Markup Language
WS:	Web Service

SUMÁRIO

1 INTRODUÇÃO

Para Sommerville (2011), o mundo moderno não existiria sem o auxílio do software. Os sistemas computacionais estão presentes nos mais diferentes segmentos de mercado e de serviços, como, gestão serviços, produtos eletrônicos, entre outros. Dessa forma, além de contribuírem com os trabalhos e afazeres do dia a dia, os softwares contribuem com entretenimento e lazer, sendo fortemente utilizado em produção de música, filmes e jogos.

Conforme Schreiber et al. (2002), a Tecnologia da Informação (TI), está diretamente ligada ao grande salto na evolução do ser humano. O grande volume de informações e velocidade com que a mesma é disseminada fez com que a evolução tecnológica envolvesse todo o mundo. Essa evolução da tecnologia da informação, possibilitou que inúmeras ferramentas surgissem, as chamadas ferramentas de gestão do conhecimento, tais como: *Application Programming Interface* (*API)*; *Business Intelligence (BI)*; *Customer Relationship Management (CRM)*; *Enterprise Resources Planning (ERP)* entre tantas outras. Tudo se consolidou de uma forma que não seria possível sem a internet.

Habitualmente são encontradas organizações cujos Sistemas de Informação (SI) estão distribuídos entre diversos processos e com diversos sistemas para controles específicos, onde as integrações entre esses são feitas através de canais de comunicação. A evolução das organizações e das tecnologias faz com que haja a necessidade de integração dos SI. Algumas situações que precisam de soluções específicas são: a disponibilização externa de informação de sistemas internos, o compartilhamento de informação entre as aplicações isoladas e até a automatização de processos. Essas soluções específicas não são encontradas facilmente pelas organizações (MARTINS, 2005).

Segundo Elmasri e Navathe (2018), alguns dos componentes fundamentais para a vida da sociedade moderna são os bancos de dados e os sistemas de banco de dados, pois, o uso dessas ferramentas tecnológicas estão presentes em diversos locais no dia a dia da sociedade. O uso das bases de dados são amplamente utilizados e podem variar desde as compras no supermercado, que geram uma atualização automática dos estoques em um banco de dados, vinculado ao sistema de gerenciamento da empresa, até o uso de redes sociais, que utilizam banco de dados enormes para armazenar postagens, fotos, vídeos, entre outros dados, e em grandes quantidades. Com essas informações, os autores Elmasri e Navathe (2018), reconhecem que os bancos de dados desempenham um papel fundamental em quase todas as áreas que utilizam computadores, sejam elas comércios locais, medicina, genética, mídias sociais entre outras.

Há muito tempo que a computação existe e nesse período os bancos relacionais são utilizados constantemente, mesmo quando as tecnologias e linguagens de programação mudavam, se atualizavam, a questão debatida entre os arquitetos de software era de qual banco relacional utilizar, mesmo tendo surgido algumas tecnologias que desafiaram a esse padrão, ele se manteve estável (SADALAGE; FOWLER, 2013).

Sadalage e Fowler (2013), complementam que a estabilidade garantida por essa tipo de banco de dados é de grande valor, uma vez sabendo que os dados de uma organização podem ser necessários e utilizados por muito tempo, e por isso, é indispensável que exista uma tecnologia desse porte, que seja acessível de diversas plataformas de programação, sendo bem compreendido e estável.

Mesmo não estando bem definido o termo NoSQL geralmente é utilizado para referenciar bancos de dados não relacionais. Ele é uma alternativa ao modelo dos bancos de dados relacionais, que surgiu com a finalidade de suprir a necessidade de armazenamento de grandes volumes de dados, com flexibilidade. Esse novo padrão de banco de dados diferencia-se principalmente pela forma de armazenamento de dados, no qual os dados são gravados sem esquema. Porém não está restrito a isso, a facilidade no desenvolvimento de aplicativos, escalabilidade e melhor desempenho são fatores que são apontados pelos utilizadores (SADALAGE; FOWLER, 2013).

Sadalage e Fowler (2013), complementam que o fato do termo NoSQL estar relacionado aos bancos de dados não relacionais, foi resultado de uma reunião de 2009, e a

partir dessa reunião que nome NoSQL ganhou força e visibilidade, mas não teve uma definição exata, apresentando algumas características em banco de dados que tendem a ser chamados de bancos NoSQL. Essas características podem ser a não utilização da linguagem *SQL*, assim como o armazenamento sem uma estrutura pré-definida. Outra característica dos bancos de dados NoSQL é de que normalmente são projetos de código livre.

Sadalage e Fowler (2013), complementam que o entendimento de NoSQL é incoerentemente traduzido para *Not Only SQL*, pois nesta tradução os bancos de dados relacionais que utilizam outras linguagens, como por exemplo Oracle e PostgreSQL, se enquadram nessa definição, por isso, ao definir um banco de dados como NoSQL, é em referência ao conjunto de características apresentadas anteriormente.

A classificação desse novo tipo para banco de dados pode ser dividido em quatro categorias, conforme sua estrutura de armazenamento, que são: chave-valor, utilizados para acesso rápido, através da identificação da chave é obtido o valor, que pode ser um dado simples, um objeto, ou até algo mais complexo. Documentos, onde cada documento pode ser representado por formatos conhecidos, como por exemplo o *JSON*. Colunas, onde as tabelas são particionadas em colunas, cada família de coluna é armazenada em um determinado arquivo. Por último, grafos, onde os dados são representados, como o próprio nome diz, através de grafos, onde os nós relacionados podem ser encontrados através das arestas (ELMASRI; NAVATHE, 2018).

Assim com as estruturas dos bancos de dados relacionais e não relacionais se diferem uma da outra, o acesso aos seus dados também são divergentes, sendo facilmente identificados através dos seus comandos de busca, inserção, remoção e atualização e até mesmo na definição de atributos utilizados nessas instruções. Essas diferenças podem ser encontradas em bancos do mesmo tipo, mas de sistemas de gerenciamento de banco de dados diferentes, nesse caso a diferenciação é menos acentuada, mas relevantes no momento da construção de um software.

1.1 Motivação

Os sistemas gerenciadores de banco de dados, relacionais e não relacionas, possuem funcionalidades diferentes e específicas, onde o acesso e a manipulação dos dados varia não

só pelo tipo de banco de dados mas também pelo SGBD utilizado. Essas características distintas geram uma dificuldade para encontrar profissionais que sejam capacitados em ambos os tipos de bancos de dados. Essas dificuldades criam problemas para empresas que possuem ou desenvolvem múltiplos softwares, onde cada um deles podem utilizar um tipo de banco de dados específico.

Tendo conhecimento das dificuldades existentes é importante a criação de uma solução, que possa oferecer auxílios e facilidades para o processo de desenvolvimento, de forma que a manipulação e o acesso aos dados seja simples, rápida e eficiente, tudo isso sem a necessidade de um conhecimento específico para trabalhar com bancos de dados relacionais e não relacionais, fazendo com que esses obstáculos não sejam um empecilho na resolução dos problemas existentes.

Assim como a escassez de profissionais, a escassez de ferramentas que detalham informações sobre a utilização dos dados contidos nos bancos, servem como motivação para o desenvolvimento deste trabalho.

1.2 Objetivos

Nessa seção são abordados os objetivos do presente trabalho, sendo eles o objetivo geral e os objetivos específicos.

1.2.1 Objetivo geral

O objetivo deste trabalho consiste em desenvolver uma ferramenta que auxilie desenvolvedores e administradores de software, construir, manter e integrar aplicações. A manipulação e o acesso aos dados é padronizada e de forma homogênea através da disponibilização de uma *API REST*, sendo a sua utilização igual, tanto para um banco de dados relacional quanto para um banco de dados não relacional, deixando cada uma das suas particularidades isoladas e ocultas dos utilizadores.

1.2.2 Objetivos específicos

São objetivos específicos desse trabalho:

I – Pesquisar sobre banco de dados relacionais e não relacionais assim como *web services REST;*

II – Implementar uma ferramenta que disponibiliza uma API *REST* para banco de dados MongoDB e PostgreSQL;

III – Validar a ferramenta com profissionais da área de desenvolvimento de *software* de forma qualitativa;

IV – Analisar os resultados.

1.3 Organização do trabalho

A apresentação do presente trabalho foi dividida em capítulos e para facilitar o entendimento do mesmo como um todo, serão apresentados conforme segue.

A análise inicia-se com o CAPÍTULO 2, onde é apresentado o referencial teórico, contendo a revisão dos conceitos que serão aplicados no trabalho, que são divididos em assuntos como: Banco de dados, com foco na pesquisa dos modelos de banco de dados relacionais e não relacionais, Integração de sistemas, *Web Services* e comunicação via *REST*.

No CAPÍTULO 3 são aprestadas algumas ferramentas relacionadas com o presente trabalho, onde é realizada uma breve descrição sobre elas, mostrando as funcionalidades que se identificam com as características da solução que foi construída.

A metodologia é apresentado no CAPÍTULO 4, mostrando qual foi a metodologia utilizada no presente trabalho assim como o método científico no qual o mesmo é baseado. É nesse capítulo que é informado quais foram as tecnologias utilizadas para a concepção da ferramenta.

No CAPÍTULO 5, serão expostas as especificações do projeto, contendo o detalhamento da arquitetura da ferramenta desenvolvida, os requisitos funcionais e não

funcionais, bem como as imagens das interfaces da aplicação detalhando as suas funcionalidades.

O CAPÍTULO 6, mostra os resultados obtidos através do questionário que foi aplicado com os profissionais que atuam na área de desenvolvimento de software.

As considerações finais serão abordadas no CAPÍTULO 7, onde é possível identificar quais são as conclusões obtidas no presente trabalho bem coimo apontamentos de implementações futuras.

2 REFERENCIAL TEÓRICO

Nessa seção serão tratados conceitos sobre banco de dados, especificando as suas principais características, assim como um detalhamento maior nos modelos de SGBDs relacionais e orientados a documentos. É também nesse capítulo que será aprofundado o conceito de integrações de sistemas de informação, *Web Services*, com uma aprofundação na comunicação via *REST*. Os conceitos discutidos neste capítulo tem como objetivo embasar teoricamente o presente trabalho.

2.1 Banco de dados

Segundo Elmasri e Navathe (2018), um Banco de Dados (BD) pode ser definido como uma estrutura de armazenamento de uma coleção dados, onde o mesmo pode ser chamado de minimundo, pois ele representa alguns aspectos do mundo real. Um agrupamento de informações que contenham algum significado coeso pode ser definido como banco de dados, onde é projetado para alguma finalidade específica e que será utilizado por um grupo de usuários e em alguns softwares.

Para Date (2004, p. 3), “O banco de dados, por si só, pode ser considerado como o equivalente eletrônico de um armário de arquivamento”. Date ainda complementa que um banco de dados é um conjunto de informações operacionais, utilizados pelas empresas em seus softwares.

Existem uma série de ferramentas conceituais que auxiliam a estrutura dos BDs, cujo objetivo principal é oferecer uma forma de descrever um projeto de banco de dados. Essas ferramentas normalmente são utilizadas para descrever os dados e as suas inter-relações, seus

tipos, suas semânticas e restrições de consistências (SILBERSCHATZ; KORTH; SUDARSHAN, 2012).

A maioria dos modelos de dados também oferece um conjunto de operações e funcionalidades para detalhar recuperações e atualizações nos bancos de dados e podem ser classificados de acordo com o conceito que utilizam para descrever as suas estruturas. Conceitos como entidades, atributos e relacionamento são utilizados por esses modelos. Uma entidade pode ser explicada como um objeto do mundo real, ou um projeto do minimundo que é representado no banco de dados, como, por exemplo, um funcionário. Um atributo pode ser definido como uma propriedade que identifica uma entidade, como por exemplo o nome do funcionário. E um relacionamento é a associação entre duas ou mais entidades, como por exemplo a ligação entre um funcionário e suas atividades na empresa (ELMASRI; NAVATHE, 2018).

Para Silberschatz, Korth e Sudarshan (2012), para manipulação do conjunto de dados, que contém as informações de uma empresa, que por sua vez, também é conhecido como banco de dados, são utilizados softwares específicos para essa finalidade, chamados de Sistemas de Gerenciamento de Banco de Dados (SGBD), que dentre todas as funcionalidades que os mesmos oferecem, as principais são a de armazenar e a de recuperar informações em um banco de dados de maneira adequada e eficaz.

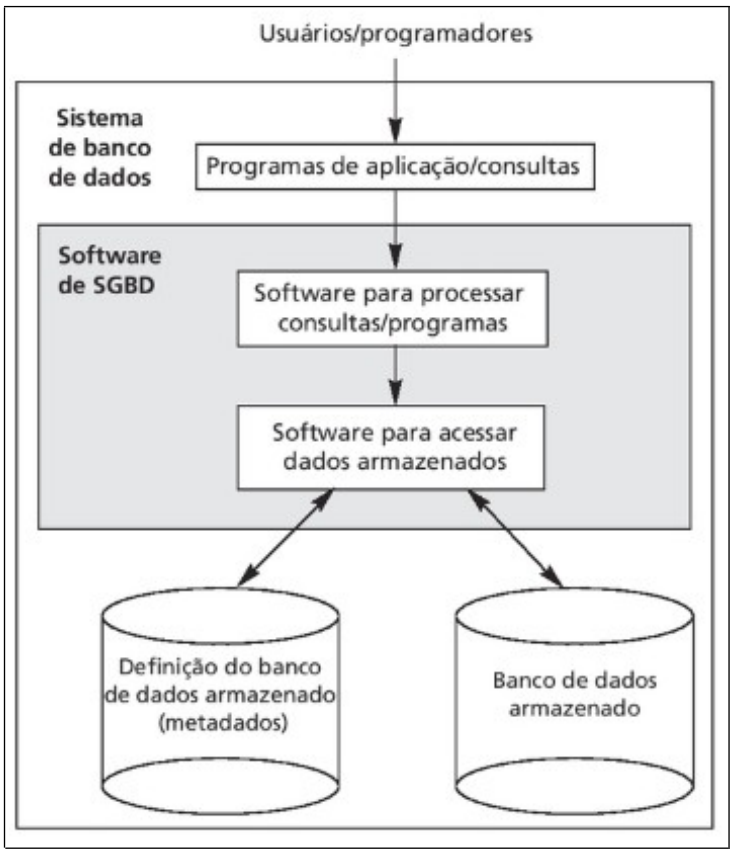
Elmasri e Navathe (2018), definem que o SGBD é um software no qual os usuários podem criar e manter um banco de dados, facilitando o processo de definição, construção, manipulação e compartilhamento entre as diversas aplicações e usuários. Onde as aplicações remetem requisições para esse software, com o objetivo de manipular o banco de dados, realizando consultas ou atualizações, como por exemplo, as ações de listar e alterar atributos de entidades assim como as de criar e remover registros, sendo todas essas instruções solicitadas através da linguagem de consultas que o SGBD oferece.

Os SGBDs podem ainda serem definidos como uma ferramenta que possui funcionalidades para recuperação, alteração e definição de dados em um determinado BD, onde cada função tem objetivo de aumentar a produtividade dos usuários que o utilizam, deixando a manipulação e manutenção dos programas mais simples e mais compreensíveis (HEUSER, 2009).

A união entre as aplicações, os bancos de dados e o SGBD é definido como sistema de banco de dados, exemplificado na Figura 1.

Segundo Vicci (2014), os SGBDs podem ser agrupados não apenas pelo modelo de dados que o mesmo utiliza, como relacional ou de rede. Esses sistemas podem ainda ser categorizados pelo número de usuários suportados, onde esses SGBDs podem ser sistemas que possibilitam apenas um usuário conectado, ou que permitem múltiplos. Outra forma de identificar esses sistemas, é pelo número de locais em que o banco de dados pode estar alocado, sendo centralizado quando os dados estão em um único ambiente, ou distribuídos quando o banco de dados fica espalhado em mais de um local, neste caso o SGBD realiza a conexão entre as partes utilizando uma rede de computadores.

Figura 1 – Exemplificação de sistema de banco de dados



Fonte: Elmasri e Navathe (2018, p. 6).

Quando categorizado pelo modelo de dados em que ele é baseado, o SGBD pode ser classificado em alguns modelos, onde o mais comum entre esses sistemas, é o relacional, mais detalhado na seção 2.1.1, que utiliza basicamente *Structured Query Language (SQL)* como linguagem. Existem diversos modelos desses sistemas, sendo alguns deles, o modelo orientado a objetos e os sistemas de armazenamento de grandes volumes de dados,

conhecidos como sistemas para armazenamento de *Big Data*, que podem ser encontrados como sistemas NoSQL, mais detalhados na seção 2.1.2. Esses modelos podem ser baseados em documentos, gráficos, colunas o em estruturas de chave-valor. Para alguns sistemas legados os modelos hierárquico, rede e *XML* ainda são utilizados (ELMASRI; NAVATHE, 2018).

2.1.1 Banco de dados relacionais

Segundo Elmasri e Navathe (2018) o modelo de dados relacional define um BD como coleções de relações, onde cada relação pode ser entendida com uma tabela, e cada linha dessa tabela pode ser compreendida como um registro com valores relacionados. Esses valores comumente descrevem um fato de um relacionamento do mundo real. Para auxiliar no entendimento dos significados dos conteúdos que são informados em cada linha, são definidos nomes tanto para a tabela quanto para suas colunas.

Para Silberschatz, Korth e Sudarshan (2012), um BD relacional é resumido em uma coleção de tabelas, também chamadas de relações ou entidades, onde cada uma delas pode conter diversas linhas com registros, essas também são conhecidas com tuplas. Cada linha guarda dados referentes a um registro da tabela. Essas informações são separadas em colunas, onde cada coluna representa um atributo da relação. Para um melhor entendimento, na Tabela 1, é possível identificar a relação de músicos, com três atributos (id, nome e estilo) e com cinco tuplas.

Reforçando o que foi especificado por Silberschatz, Korth e Sudarshan (2012) Elmasri, Navathe, (2018), resumem:

> Na terminologia formal do modelo relacional, uma linha é chamada de tupla, um cabeçalho da coluna é chamado de atributo e a tabela é chamada de relação. O tipo de dado que descreve os tipos de valores que podem aparecer em cada coluna é representado por um domínio de valores possíveis (Elmasri; Navathe, 2018, p. 136).

Segundo Elmasri e Navathe (2018), um domínio é um agrupamento de valores atômicos que descrevem um atributo, sendo possível a divisão dele entre duas partes. A primeira pode ser chamada de domínio lógico onde é definido um padrão para os dados que serão informados naquela coluna para todas as linhas, como por exemplo, em uma tabela de alunos de uma universidade o atributo *"nome"* pode ser representado por uma cadeia de

caracteres, já o atributo *"cpf"* pode ser representado por onze dígitos e o atributo *"idade"* por ser definido pelas idades possíveis dos alunos existentes na universidade; por exemplo um valor inteiro entre quinze e oitenta. A segunda parte é a definição de um tipo de dado; por exemplo o atributo *"nome"* seria do tipo *string*, *"cpf"* seria também do *string* e o atributo *"idade"* do tipo inteiro. Com essa definição a interpretação de cada valor nas colunas de uma tupla pode ser feita através do cabeçalho que contém as informações dos domínios. Em tuplas cujo valor dos atributos ainda são desconhecidos ou inexistentes o valor da coluna é *NULL*, veja um exemplo de tuplas na Tabela 1.

Tabela 1 – Exemplo de uma relação com tuplas e atributos

id	**nome**	**estilo**
10101	Lucas	RAP
10102	José	SOUL
10106	Paula	BLUES
10110	Patrícia	HIP-HOP
09109	Maria	NULL

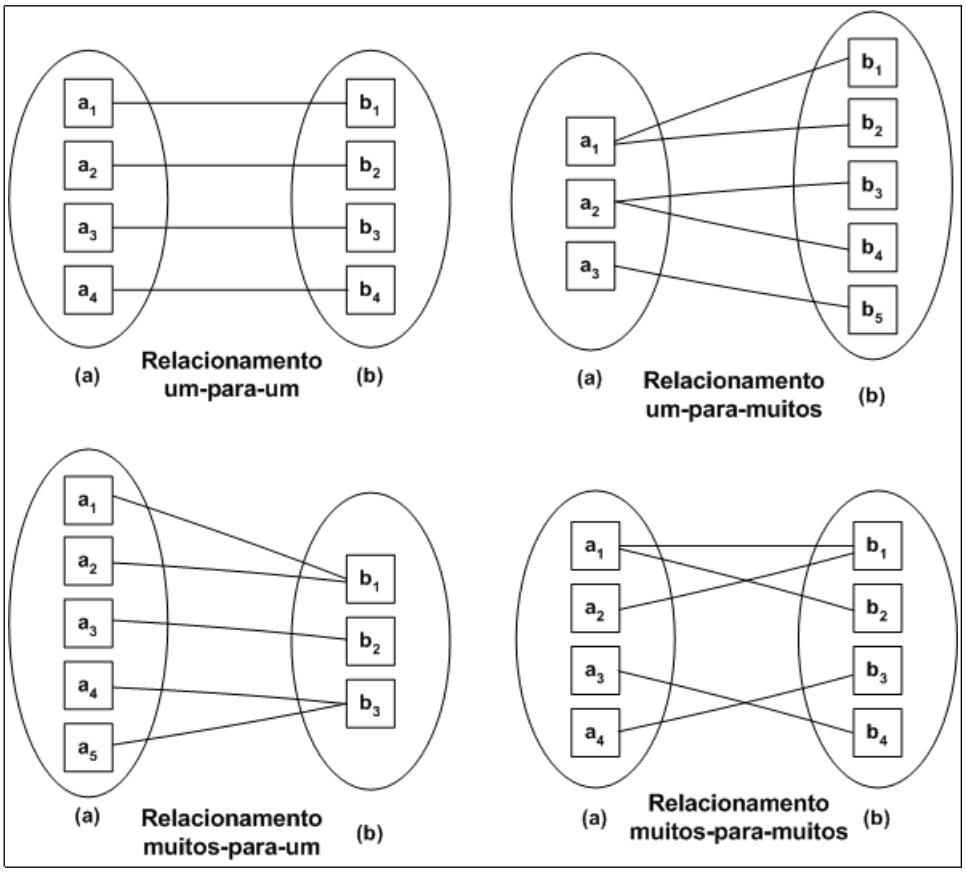
Fonte: Do autor, adaptado de Silberschatz, Korth e Sudarshan (2012).

Silberschatz, Korth e Sudarshan (1999), definem que um banco de dados pode conter um conjunto de entidades com um conjunto de relacionamentos, o que acaba evitando o armazenamento de diversos dados repetidos. Através da criação de referências em vez de fazer uma cópia dos dados toda a vez que é necessária a ligação entre duas entidades a atualização dos registros acaba se tornando muito mais simples. Para a construção desses relacionamentos é preciso estabelecer algumas restrições de dependências e de cardinalidades.

Cardinalidade são os números de relações em que uma entidade pode estar associada, sendo úteis para a descrição dos conjuntos de relacionamentos. Essas relações são divididas em quatro tipos, onde pelo menos um dos tipos deve ser seguido ao ser efetuada uma ligação entre duas entidades, como pode ser visto na Figura 2. O primeiro tipo é definido em uma relação um para um (1:1), onde um registro da relação *(a)* pode estar associada somente a um registro de uma relação *(b)*, e um registro da relação (*b)*, pode estar relacionado a somente um relacionamento da relação (*a)*. Seguindo temos o tipo um para muitos (1:n), onde a entidade (*a)* está associada a diversos registros em (*b)*, mas (*b)* deve estar associada a somente a um registro de (*a)*. Muitos para um (n:1) é o relacionamento onde muitos registros da entidade *(a)*

estão associados a muitos registros da entidade (*b)* em contrapartida um registro da entidade *(b)* pode estar relacionada a um registro da entidade *(a)*. E ainda temos o tipo muitos para muitos (n:n) onde uma entidade *(a)* pode estar associada a quantos registros da entidade *(b)*, e vice-versa também um registro da entidade *(b)* pode estar vinculada a um número indefinido da tabela *(a)* (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

Figura 2 – Tipos de cardinalidades



Fonte: Brito (2013, texto digital).

Para Silberschatz, Korth e Sudarshan (1999) outro fator fundamental para o entendimento dos bancos de dados relacionais é a dependência de existência, onde são definidos entidades dominantes e dominadas, fazendo com que a entidade dominada possa existir somente em conjunto da entidade dominante. Essa restrição torna os dados entre elas amarrados de maneira que ao excluir um dado da entidade dominante os seus relacionados que estão na entidade dominada sejam excluídos também. A identificação desses relacionamentos é feita através de atributos especiais chamados de chaves, que podem ser simples, onde apenas um atributo identifica a relação, ou compostos, onde existe um conjunto de atributos para representar a ligação entre as entidades relacionadas, neste segundo caso a restrição pode ser chamada de superchave. O atributo que será escolhido como chave da

entidade deve ser escolhido como o objetivo de identificar um registro da tabela de forma única, representando uma restrição do mundo real da empresa que está sendo modelada, esse atributo pode ser identificado como chave primária, podendo ser simples ou composto.

Elmasri e Navathe (2018), complementam que nenhuma chave primária pode ser nula, porque ela serve para a identificação das tuplas, para que nenhuma seja igual todos os registros devem conter um valor diferente. Para a definição da integridade de relacionamento entre duas entidades, é estabelecido o conceito de chaves estrangeiras, onde a mesma deve possuir o mesmo domínio da chave primária da qual ela faz referência.

Após um breve entendimento sobre relações, tuplas e atributos, assim como seus relacionamentos e suas restrições, será abordado no tópico 2.1.2, o que Elmasri e Navathe (2018) definem como a linguagem padrão dos SGBDs, a *SQL*.

2.1.1.1 Structured Query Language (SQL)

A linguagem *SQL* possui diversos recursos que vão além das funcionalidades de consulta ao banco de dados, ela pode ser utilizada para a definição de estruturas, modificações de dados e para a especificação de segurança, sendo dividida em duas partes. A *Data Definition Language (DDL)* é a linguagem de definição de dados, utilizada para definição e modificação de esquemas de relações, criação e remoção de índices. A *Data Manipulation Language (DML)* é a parte da *SQL* que abrange as consultas e as operações de inserção, exclusão e modificação de linhas no BD (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

2.1.1.1.1 Data Definition Language (DDL)

A *SQL DDL* é utilizada para especificar um conjunto de relações assim como as suas propriedades, sejam elas as regras de integridade ou os domínios dos valores de cada atributo. O comando *CREATE* é um dos que compõem essa linguagem, ele é utilizado para a criação de uma nova estrutura, como indexes, domínios, relacionamentos e relações. Na criação de tabelas o comando é utilizado junto com a especificação do nome da relação e de seus atributos com os respectivos domínios e suas restrições e dependências como é exibido nas

Figuras 3 e 4, onde mostram respectivamente a entidade de funcionário e de dependente, junto com seus atributos, domínios e chaves primárias e estrangeiras. (ELMASRI; NAVATHE, 2018).

Segundo Elmasri e Navathe (2018), essa linguagem fornece outros comandos de alteração da base de dados, como o *DROP*, que pode remover todos os elementos nomeados do BD, como tabelas, domínios, restrições e tipos; e o comando *ALTER*, que pode alterar os elementos nomeados assim como o comando *DROP*, em tabelas essa operação também pode manipular as colunas e as restrições, alterando, renomeando, incluindo e removendo.

Figura 3 – Exemplo de criação de uma tabela utilizando *SQL*

```
CREATE TABLE FUNCIONARIO
    (Primeiro_nome          VARCHAR(15)         NOT NULL,
    Nome_meio               CHAR,
    Ultimo_nome             VARCHAR(15)         NOT NULL,
    Cpf                     CHAR(11),           NOT NULL,
    Data_nascimento         DATE,
    Endereco                VARCHAR(30),
    Sexo                    CHAR,
    Salario                 DECIMAL(10,2),
    Cpf_supervisor          CHAR(11),
    Numero_departamento     INT                 NOT NULL,
    PRIMARY KEY (Cpf) );
```

Fonte: Elmasri, Navathe (2018, p. 162).

Figura 4 – Exemplo de criação de uma tabela utilizando *SQL*, com chave estrangeira

```
CREATE TABLE DEPENDENTE
    (Cpf_funcionario        CHAR(11),           NOT NULL,
    Nome_dependente         VARCHAR(15)         NOT NULL,
    Sexo                    CHAR,
    Data_nascimento         DATE,
    Parentesco                                  VARCHAR(8),
    PRIMARY KEY (Cpf_funcionario, Nome_dependente),
    FOREIGN KEY (Cpf_funcionario) REFERENCES FUNCIONARIO(Cpf) );
```

Fonte: Elmasri, Navathe (2018, p. 162).

Para a criação das colunas existem alguns domínios pré-estabelecidos pelos bancos de dados relacionais, que podem ser divididos em alguns grupos. Os numéricos, que são *INT*, *SMALLINT*, *FLOAT*, *DOUBLE PRECISION*, *DECIMAL* e *NUMERIC*. Os de cadeia de caracteres, que são *CHAR(n)*, *VARCHAR(n)*, *TEXT* e *CLOB*. Os de sequência de bits, que são *BIT(n)*, *BIT VARYING(n)*. Os tipos *booleanos*, que podem assumir os valores de *TRUE* e

FALSE. Os relacionados a tempo, que são *DATE, TIMESTAMP e TIME*. Além desses tipos é possível a criação de outros conforme a necessidade dos usuários, através do comando da *DDL*, *CREATE TYPE* (ELMASRI; NAVATHE, 2018).

2.1.1.1.2 Data Manipulation Language (DML)

A linguagem de manipulação de dados, serve tanto para a realização de busca de dados quanto para a inserção, atualização e remoção de registros de uma relação de um banco de dados (VICCI, 2014).

A realização das consultas em *SQL*, são divididas em três componentes básicos, *SELECT*, *FROM* e *WHERE*, com o seguinte formato: *"SELECT <lista de atributos> FROM <lista de tabelas> WHERE <condições>;"*. A lista de atributos na expressão anterior, é a lista das colunas que devem ser recuperados pela consulta. A lista de tabelas, são as relações das quais devem ser buscados os atributos. As condições, são as expressões condicionais que descrevem a regra para definir quais linhas das relações devem ser recuperadas do banco de dados. A seguir na Figura 5, pode-se visualizar um comando de busca (ELMASRI; NAVATHE, 2018).

Figura 5 – Exemplo de busca de dados utilizando *SQL*

```
SELECT Data_nascimento, Endereco
FROM   FUNCIONARIO
WHERE  Primeiro_nome='João' AND Nome_meio='B' AND
       Ultimo_nome='Silva';
```

Fonte: Elmasri, Navathe (2018, p. 170).

Elmasri e Navathe (2018), afirmam que a linguagem *SQL*, possui três comandos em que é possível efetuar mudanças no banco de dados, que são, *INSERT*, *DELETE* e *UPDATE*. O *INSERT* é utilizado para inserir novas linhas em uma relação, respeitando todas as regras de restrições e domínios contidos nela. Para a execução desse comando é necessária a especificação do nome da relação e os valores a serem adicionados, o nome dos atributos é opcional, porém ao não informar será considerado obrigatório identificar valores para todos os campos da tabela, para melhor entendimento veja as Figuras 6 e 7 que mostram uma inserção com atributos específicos e outra sem, respectivamente.

Figura 6 – Exemplo de *INSERT*, sem identificar atributos utilizando *SQL*

```
INSERT INTO   FUNCIONARIO
VALUES        ('Ricardo', 'K', 'Marini', '65329865388', '30-12-1962',
              'Rua Itapira, 44, Santos, SP', 'M', 37000, '65329865388', 4 );
```

Fonte: Elmasri, Navathe (2018, p. 181).

Figura 7 – Exemplo de *INSERT*, identificando atributos utilizando *SQL*

```
INSERT INTO   FUNCIONARIO (Primeiro_nome, Ultimo_nome, Numero_
              departamento, Cpf)
VALUES        ('Ricardo', 'Marini', 4, '65329865388');
```

Fonte: Elmasri, Navathe (2018, p. 181).

Como é apresentado na Figura 8, o comando *DELETE* é utilizado para a remoção de linhas em uma tabela, para a execução desse comando, assim como para o *INSERT*, é necessário a informação do nome da relação e uma cláusula adicional chamada *WHERE* opcional, que caso seja omitida, todos os registros da tabela serão deletados, se informado as remoções serão somente dos registros que satisfaçam a condição definida. Em ambos os casos as restrições definidas na tabela devem ser respeitadas (ELMASRI; NAVATHE, 2018).

Figura 8 – Exemplo de *DELETE* utilizando *SQL*

```
DELETE FROM   FUNCIONARIO
WHERE         Ultimo_nome = 'Braga';
```

Fonte: Elmasri, Navathe (2018, p. 182)

Segundo Vicci (2014), a instrução *UPDATE* é utilizada para alterar os valores de uma ou mais linha de uma entidade, e da mesma forma que o comando *DELETE*, ele também precisa indicar o nome da entidade em que os dados serão alterados e da mesma forma, pode ou não conter a cláusula *WHERE*, e da mesma maneira, caso não especificado, todos os registros contidos na tabela serão atualizados. Ainda para esse comando é necessário o acréscimo da cláusula *SET*, que recebe o nome dos atributos que deverão ser alterados e seus novos valores, como é exibido na Figura 9.

Figura 9 – Exemplo de *UPDATE*

```
UPDATE        PROJETO
SET           Projlocal = 'Recife', Dnum
              = 5
WHERE         Projnumero=10;
```

Fonte: Vice (2014, p. 155).

Atualmente existem diversos tipos de BD relacionais que suportam a linguagem *SQL*, dentre eles alguns são o MySQL, PostgreSQL, Oracle e o SQLServer (ELMASRI; NAVATHE, 2018).

2.1.2 Banco de dados não relacionais

A solução comumente chamada de NoSQL, trouxe mudanças para os bancos de dados em relação ao modelo relacional que já estava bem-sucedido e dominante no mercado. Cada solução baseada nesse novo conceito tem suas particularidades, podendo ser divididas em quatro principais categorias, chave-valor, documentos, colunas e grafos. Dessas categorias que são amplamente utilizadas no ecossistema de NoSQL, com exceção da categoria de grafos, aplicam em seus modelos de dados um conceito denominado de orientação agregada (SADALAGE; FOWLER, 2013).

Sadalage e Fowler (2013), definem que o modelo de dados relacional possui uma estrutura limitada, onde é impossível inserir uma lista de registros ou de valores vinculados a outras estruturas, e nem realizar consultas aninhadas ou inserir dados aninhados. Esse ponto difere dos bancos que utilizam a orientação agregada, que é caracterizada como:

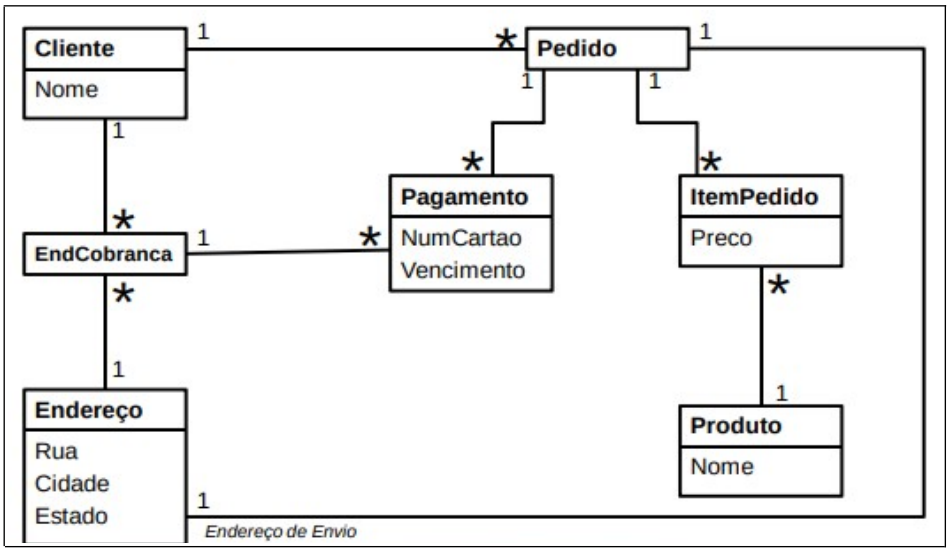
> A orientação agregada utiliza uma abordagem diferente. Ela reconhece que você, frequentemente, deseja trabalhar com os dados na forma de unidades que tenham uma estrutura mais complexa do que um conjunto de tuplas. Pode ser útil pensar em termos de um registro complexo que permita que listas e outras estruturas de dados sejam aninhadas dentro dele (SADALAGE; FOWLER, 2013, p. 42).

Um exemplo com as diferenças pode ajudar na explicação e no entendimento. Para isso podem ser analisadas as imagens que seguem, nas Figuras 10 e 11, onde é possível visualizar, respectivamente, um exemplo de modelagem relacional e o mesmo banco configurado em um modelo de dados agregado. Na Figura 12, é possível ver o resultado da modelagem agregada em um formato de *JavaScript Object Notation (JSON)*, que é comumente usado para representar os dados em NoSQL (SADALAGE; FOWLER, 2013).

Na Figura 10, é possível verificar que nenhum dado é repetido nas tabelas, já que a modelagem está normalizada, e possui as devidas integridades referenciais. Já na Figura 11, temos dois agregados principais, cliente e pedido. Na agregação de cliente existem uma lista de endereços e a agregação do pedido possui uma lista de itens requisitados. Como é

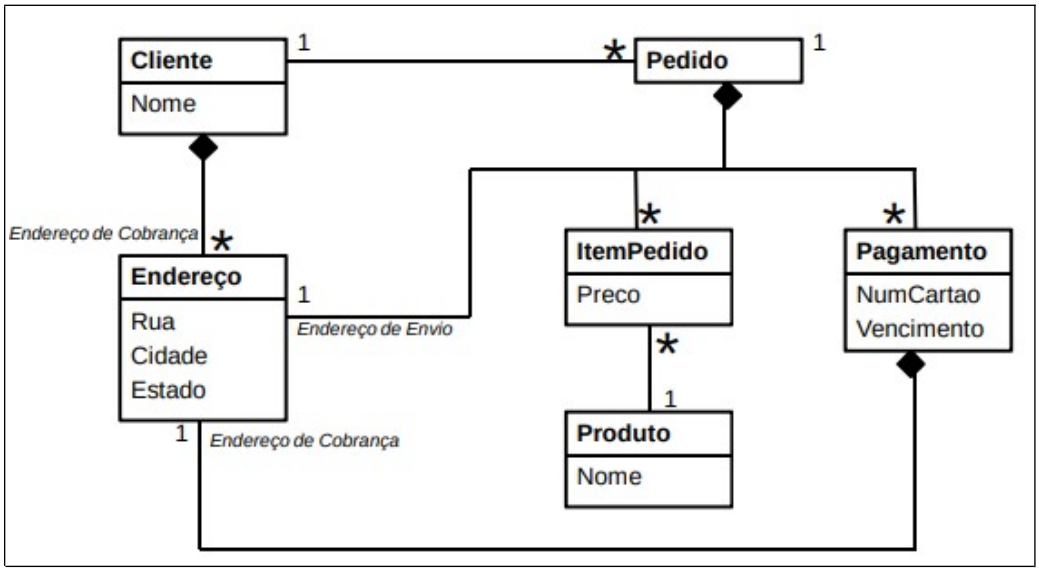
mostrado na Figura 12, o modelo representado na Figura 11 gera cópias dos dados para as estruturas que estão sendo agregadas, como é exibido no exemplo da Figura 12, onde o registro de endereço aparece três vezes, que ao contrário dos modelos relacionais normalizados, onde seriam utilizadas referências para outras tabelas, nesse caso ele é considerado como um valor, sendo copiado toda a cada vez que é utilizado (SADALAGE; FOWLER, 2013).

Figura 10 – Exemplo de modelo de dados relacional



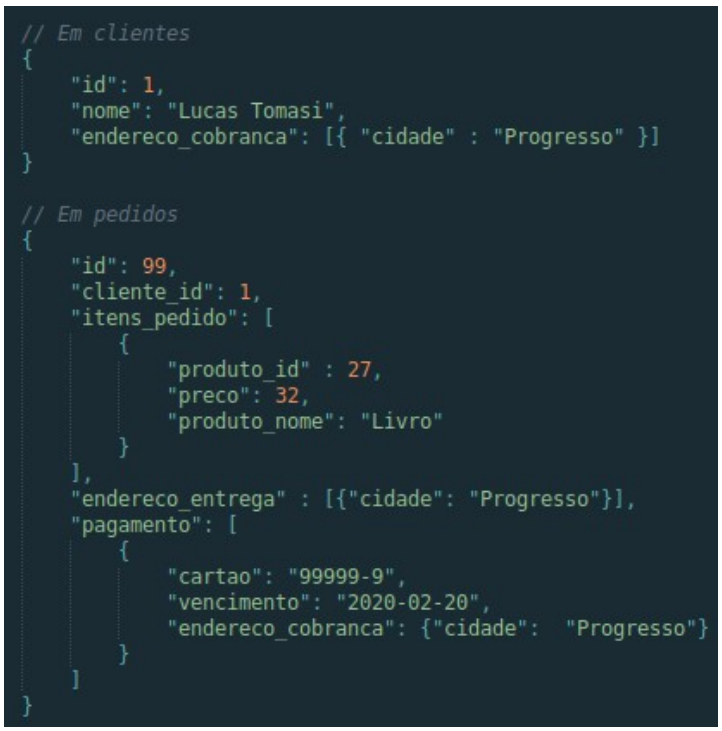
Fonte: Do autor, adaptado de Sadalage, Fowler (2013, p. 43).

Figura 11 – Exemplo de modelo de dados agregado



Fonte: Do autor, adaptado de Sadalage, Fowler (2013, p. 45).

Figura 12 – Representação do modelo agregado em *JSON*



```
// Em clientes
{
    "id": 1,
    "nome": "Lucas Tomasi",
    "endereco_cobranca": [{ "cidade" : "Progresso" }]
}

// Em pedidos
{
    "id": 99,
    "cliente_id": 1,
    "itens_pedido": [
        {
            "produto_id" : 27,
            "preco": 32,
            "produto_nome": "Livro"
        }
    ],
    "endereco_entrega" : [{"cidade": "Progresso"}],
    "pagamento": [
        {
            "cartao": "99999-9",
            "vencimento": "2020-02-20",
            "endereco_cobranca": {"cidade":  "Progresso"}
        }
    ]
}
```

Fonte: Do autor, adaptado de Sadalage, Fowler (2013, p. 45).

Nesse tipo de modelagem se faz necessário utilizar um pensamento voltado ao acesso dos dados e de como eles serão utilizados posteriormente, pois a definição dos limites de agregação pode variar de acordo com o cenário proposto. Por exemplo, uma outra agregação possível nessa hipótese, é de pedidos com clientes, onde o cliente possuiria uma lista de pedidos. Visto isso, é possível perceber que não existe uma resposta global para as agregações, dependendo inteiramente dos contextos em que os dados serão utilizados (SADALAGE; FOWLER, 2013).

2.1.2.1 Banco de dados orientados a documentos

Nos BDs não relacionais baseados em documentos, o principal objetivo é armazenar e recuperar documentos, independentemente do seu tipo, podendo ser no formato *JSON*, *XML*, *BSON*, entre outros. Esses documentos podem conter mapas, coleções e valores escalares em sua estrutura hierárquica. Algumas diferenças existem em relação aos modelos relacionais, pois onde em um banco relacional possui tabelas e linhas, nos orientados a documentos possuem coleções e documentos (SADALAGE; FOWLER, 2013).

Sadalage e Fowler (2013), complementam que os documentos armazenados em uma coleção são semelhantes entre si, mas não possuem a obrigatoriedade de serem idênticos, podendo existir documentos com diferentes atributos dentro de uma mesma coleção.

Nos bancos de dados orientados a documentos são utilizadas algumas técnicas diferentes das quais são utilizadas no modelo relacional, enquanto em bases relacionais a normalização é frequentemente usada, nos modelos orientados a documentos o pensamento é outro, sendo comum a utilização da desnormalização dos dados, onde é possível encontrar diversos dados repetidos entre os documentos de uma base de dados (PANIZ, 2016).

Para as buscas de dados onde se faz necessária a obtenção de uma listagem com informações que estão gravadas em diversas coleções, seriam realizadas diversas ligações entre as coleções até conseguir identificar os dados necessários para a exibição da listagem, o que acaba gerando um custo de processamento. Com a desnormalização dos dados evita-se a utilização dessas diversas ligações, uma vez que os dados necessários estarão em um único documento. Essa forma de modelar causa duplicidade em diversos dados, porém aumenta a eficácia das buscas e inserção de dados (PANIZ, 2016).

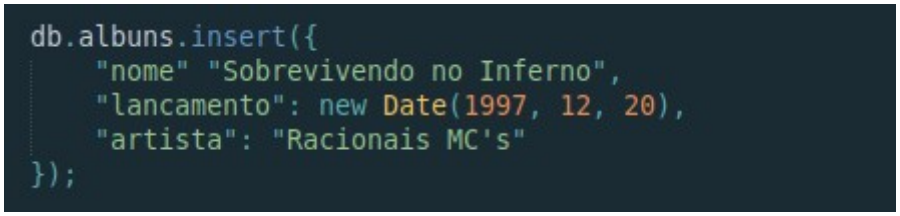
Paniz (2016), sugere que em documentos com um grande número de atributos, onde uma desnormalização resultaria em muitos dados duplicados, a referência entre documentos pode ser avaliada, e através dessa alternativa, somente alguns dados, os que são necessários, ou mais utilizados, seriam introduzidos no documento, os outros seriam buscados no documento de origem através da referência que para esses casos, também deverá ser criada, porém esse processamento seria realizado somente quando fosse necessário.

Segundo Sadalage e Fowler (2013), existem alguns bancos de dados orientados a documentos e dentre os mais populares são o MongoDB, CouchDB, Lotus Notes, RavenDB.

Para os modelos de bancos de dados orientados a documentos não existe um padrão para as características que podem existir nesse tipo de solução, podendo haver divergências entre os diversos modelos de SGBDs. Em bancos de dados criados a partir do sistema MongoDB, é possível realizar inserções e consultas através das operações fornecidas pela ferramenta, dessas operações não necessariamente são as mesmas utilizadas pelos outros sistemas NoSQL (SADALAGE; FOWLER, 2013).

No MongoDB, as operações são realizadas através de comandos JavaScript, e sempre partindo do objeto raiz, que é o *"db"*, e é através desse objeto que a manipulação dos dados serão executados. Para a inserção de documento é necessário utilizar a função de *"insert"*, que está disponível a partir das coleções do objeto *"db"*, e deve ser chamado na ordem, *"db"* nome da coleção e função, informando o documento que deve ser inserido no banco de dados, como por exemplo *"db.nome_da_colecao.insert(documento)"*, exemplificado na Figura 13 (PAINZ, 2016).

Figura 13 – Exemplo de inserção de um documento no MongoDB



Fonte: Do autor, adaptado de Painz (2016, p 15).

Diferente de BD relacionais, que necessitem de uma definição de tabelas e colunas, no MongoDB isso não se faz necessário, pois, ao utilizar uma coleção que ainda não está definida a mesma é automaticamente criada, como pode ser visto no exemplo da Figura 13, que está sendo inserido um documento na coleção de "álbuns", caso essa coleção não exista, o MongoDB cria ela e insere o documento nela (PAINZ, 2016).

Painz (2016), complementa que toda vez que é inserido um documento no MongoDB, é criado de forma automática uma *primary key* para o documento, o campo "_id", que é gerenciado pelo próprio sistema, que define um valor do tipo "ObjectId", que nada mais é que um *BSON* de 12 bytes, como pode ser visto no exemplo da Figura 14, no qual o identificador gerado foi "3179hdakdh123131lk223123".

Na Figura 14, está sendo exemplificado uma busca de um documento, que Painz (2016), explica que o banco MongoDB, oferece duas funções de busca, e como na operação de inserção, a busca recebe um objeto *JSON* como parâmetro, porém nesse caso ele é utilizado para definir um critério de filtro. As funções de busca são: *"find"*, que busca todos os registros que satisfaçam a condição definida e o *"findOne"*, que retorna o primeiro registro que é encontrado a partir do critério informado.

Figura 14 – Exemplos de consultas de documentos no MongoDB

```
db.albuns.findOne({"nome": "Sobrevivendo no Inferno"});
// retorno
{
    "_id": ObjectId("3179hdakdh123131lk223123"),
    "nome": "Sobrevivendo no Inferno",
    "lancamento": new Date(1997, 12, 20),
    "artista": "Racionais MC's"
}

db.albuns.find({"artista": "Racionais MC's"});
// retorno
{
    "_id": ObjectId("3179hdakdh123131lk223123"),
    "nome": "Sobrevivendo no Inferno",
    "lancamento": new Date(1997, 12, 20),
    "artista": "Racionais MC's"
},
{
    "_id": ObjectId("831209das8108d01n18dn018"),
    "nome": "Cores e valores",
    "lancamento": new Date(2014, 11, 25),
    "artista": "Racionais MC's"
}
```

Fonte: Do autor, adaptado de Painz (2016, p. 19).

Assim como as operações de consulta, o MongoDB oferece a funcionalidade de remoção de documentos, que também possui uma passagem de parâmetros que define o critério para remoção. Outra funcionalidade existente no MongoDB é a de alteração de documentos, que diferente das funções vistas até agora, recebe dois parâmetros. O primeiro parâmetro é o filtro, onde é informado o critério de identificação dos documentos para alteração. Da mesma maneira que o anterior, esse parâmetro também é um *JSON*, porém nesse a informação é a de quais campos serão alterados e quais serão os novos valores, exemplificado na Figura 15 (PAINZ, 2016).

Figura 15 – Exemplos de alteração de documentos no MongoDB

```
db.albuns.update({
    "_id" : ObjectId("831209das8108d01n18dn018")
}, {
    $set : {
        "nome": "Cores e Valores",
    }
});
```

Fonte: Do autor, adaptado de Painz (2016, p 27).

Sadalage e Fowler (2013), definem que a escolha da utilização de um banco de dados NoSQL deve ser baseada em alguns fatores. Primeiramente a necessidade de desempenho

para acessar dados em grandes quantidades, já que a leitura e recuperação dos dados que já estão agregados torna o processo muito mais eficaz. A produtividade do programador, já que as informações são exibidas em termos de agregados, porém essas informações necessitam ser transformadas em relações. A dificuldade de obter um esquema concreto também é um ponto forte de bases NoSQL, pois oferecem um armazenamento sem esquemas podendo ser inseridos dados não uniformes.

Sadalage e Fowler (2013), complementam ainda que existem muitas situações onde, de fato, a melhor escolha é a permanência em banco de dados do tipo relacional, sendo a possível encontrar profissionais com experiência, além disso, esses sistemas são mais consolidados, sendo menos provável encontrar partes ainda pouco testadas que uma nova tecnologia oferece. Outro fator importante é que os sistemas relacionais contam com uma gama grande de ferramentas já criadas que estão a disponibilidade dos programadores.

Para justificar a escolha de um banco NoSQL, o mesmo deve trazer alguma vantagem significativa em relação aos bancos relacionais na situação em que ele será utilizado. A utilização de bancos de dados NoSQL em muitos casos é mais vantajosa, mas isso não significa que para todos os casos e nem mesmo para a maioria ele será uma solução mais benéfica (SADALAGE; FOWLER, 2013).

2.2 Integração de sistemas

As coletas, recuperações, processamentos, armazenamentos e a distribuição de informações são possibilitados pelos sistemas de informações, facilitando o processo de controle e planejamento assim como a análise e o processo decisório da empresa (CORREIA, 2015).

Para Duarte (2009), a integração entre os sistemas de informação é de extrema importância para a maioria das empresas. Com o crescimento do número de sistemas institucionais e a utilização de sistemas legados, que são os sistemas antigos que por algum motivo específico ainda são utilizados nas empresas, a integração entre esses sistemas tornou-se indispensável, tanto para a obtenção e análise de dados, quanto para a realização dos processos da empresa.

Comumente são encontradas empresas cujos Sistemas de Informação (SI) estão distribuídos entre diversos processos e com diversos sistemas para controles específicos, onde as integrações entre esses são feitas através de canais de comunicação. A evolução das organizações e das tecnologias faz com que haja a necessidade de integração dos SI. Algumas situações que precisam de soluções específicas são: a disponibilização externa de informação de sistemas internos, o compartilhamento de informação entre as aplicações isoladas e até a automatização de processos. Essas soluções específicas não são encontradas facilmente pelas organizações (MARTINS, 2005).

Martins (2005) explica que a integração dos SI pode mostrar-se muito complexa, tanto no seu entendimento quando na sua implementação, quando a solução adotada é construída diretamente no nível de dados, como é o exemplo das integrações que utilizam a conexão direta aos SGBDs, que por sua vez, acabam não fornecendo um conjunto de funcionalidades síncrona ou assíncrona, para que a troca de informação seja entendida por sistemas heterogêneos.

A necessidade de obtenção de dados entre os diversos sistemas de informação existentes nas empresas gera um desafio no ponto de vista de integração. Esses dados são um fator importante para a integração entre sistemas (CORREIA, 2015). Duarte (2009), ainda afirma que a necessidade de integrar sistemas distintos, tanto em relação a processos quanto em relação a dados, faz com que novas formas de desenvolver aplicações surjam.

O *web service* é uma das maneiras de desenvolver aplicações pensando nas possibilidades de integração, como é mostrado na próxima seção.

2.3 *Web Services*

Web Service (WS) é um instrumento que possibilita a comunicação de diferentes aplicações, sua essência é a capacidade de tornar sistemas independentes. Fornecendo informações de forma padronizada e com estruturas flexíveis adequando-se ao ambiente proposto por cada cliente, tornando-se uma arquitetura mundialmente utilizada (SOMMERVILLE, 2011).

Duarte (2014), define *web services* como:

> Os Web Services são aplicações modulares que podem ser descritas, publicadas e invocadas sobre uma rede, geralmente a Web. Em outras palavras, é uma interface que descreve uma coleção de operações que são acessíveis pela rede através de mensagens em formato XML padronizadas. Sua estrutura arquitetura permite a comunicação entre aplicações, assim, um serviço pode ser invocado remotamente, ou ser utilizado para compor um novo serviço juntamente com outros (DUARTE, 2014, p. 15).

Segundo Kuehne (2009), o objetivo principal de um *web service* é encapsular funcionalidades que estão disponíveis remotamente, onde a resposta para a chamada será trazida por um protocolo, como o *HyperText Transfer Protocol (HTTP)*. Essas requisições podem ser feitas através um navegador para a conectar um cliente e um servidor. Segundo Gomes e Jandl (2009), essas comunicações podem resultar em diversos códigos de retorno, onde os principais podem ser vistos na Tabela 2.

Um serviço *web* pode ser criado totalmente do zero, específico para resolver um problema, ou ser transformado a partir de uma aplicação existente e até mesmo da junção de outros *web services* já existentes (KUEHNE, 2009).

A construção de um *WS* pode ser equiparada ao desenvolvimento de um componente de software, como por exemplo, um componente caixa preta, ou *black box*, do qual pode ser reutilizada as funcionalidades, sem ter a necessidade se conhecer detalhes da sua implementação (DUARTE, 2014).

Tabela 2 – Códigos de retorno *HTTP* e seus significados

Código	**Significado**
200	Sucesso
201	Criado
204	Sem conteúdo
400	Requisição inadequada
401	Não autorizado
403	Acesso negado
404	Não encontrado

Código	Significado
412	Falha na pré-condição
500	Erro interno no servidor
501	Não implementado

Do autor, adaptado de Gomes e Jandl (2009, p. 101).

Complementando Duarte (2014), Correia (2015) afirma que:

> Usuários de Web Services não precisam conhecer sua estrutura ou sua linguagem de programação para poder fazer uso dos serviços oferecidos. Estes usuários somente precisam saber do que necessita, para realizar a solicitação do serviço desejado e obter a resposta sobre o serviço solicitado (CORREIA, 2015, p. 25).

Kuehne (2009), ainda descreve que o ciclo de vida de um *WS* pode ser definido em quatro principais fases, e em cada uma delas é possível identificar requisitos específicos para desenvolver o seu ciclo de vida. A primeira fase é a construção, que se trata da fase em que inclui o desenvolvimento e a realização dos testes do *WS*, assim como as definições de interface e descrições de implementação do serviço. A segunda fase é a publicação, que é a parte onde é disponibilizado para o consumidor a descrição de implementação e a interface de utilização do serviço. É nessa parte que ocorre a publicação da parte funcional do *WS*, geralmente em um provedor de serviços. Em seguida vem a fase de execução, onde ocorre a utilização do serviço, que já foi publicado e está disponível, por meio da rede, para ser consumido. Por último existe a fase de gerenciamento, que é a fase onde ocorre o gerenciamento administrativo do *WS*, considerando requisitos como segurança, disponibilidade, desempenho, qualidade do serviço e processos de negócio.

Segundo Duarte (2014), com o crescimento notável dos SI, assim como o aumento da sua complexidade, faz com que as arquiteturas tradicionais alcancem os seus limites, tornando necessária uma nova arquitetura que possibilite o reúso dos serviços maximizando a flexibilidade.

2.3.1 Tipos de dados na comunicação via *WS*

Todo o retorno de uma requisição apresenta a informação de *Media Type*[1], que serve para indicar ao navegador qual é o tipo de informação está sendo transferida, para que ele possa interpretar o retorno de maneira adequada. Em *web services* são utilizados vários tipos de *Media Types* comumente conhecido com *Multipurpose Internet Mail Extensions (MIME)*, dentre eles estão o *text/xml*, conhecido como *eXtensible Markup Language (XML)* e o *application/json*, conhecido como *JSON* (SAUDATE, 2014).

O *JavaScript Object Notation (JSON)*, assim como o *XML*, é um tipo de dado que também pode ser definido como uma linguagem de marcação. Um *JSON* obedece ao seguinte modelo: um objeto possui zero ou mais membros; um membro possui zero ou mais pares e zero ou mais membros; um par possui uma chave e um valor; e ainda um membro pode ser um *array*[2] (SAUDATE, 2014).

2.4 Comunicação via *REST*

O estilo *REpresentational State Transfer (REST)* ao contrário dos outros protocolos como o *Simple Object Access Protocol (SOAP)*, é mais uma filosofia, um conjunto de princípios, com ideias sobre como os dados podem ser transmitidos com elegância, utilizando dos recursos do protocolo *HTTP*, como cabeçalho e verbos de requisição (MITCHELL, 2013).

Para Saudate (2014), o *REST* é um estilo de desenvolvimento de *web services*, que teve o seu surgimento através de Roy Fielding que em sua tese de doutorado apresentou o estilo, ele por sua vez também é coautor do protocolo *HTTP*, ainda nesse estilo identifica-se a utilização da boa prática do *HTTP*, que são o uso adequado dos métodos *HTTP* e de *Uniform Resource Locator (URL)*, utilização de códigos padronizados para exibir falhas e sucessos,

[1] [...].Media Types são formas padronizadas de descrever uma determinada informação. Os Media Types são divididos em tipos e subtipos, e acrescidos de parâmetros (se houverem). São compostos com o seguinte formato: tipo/subtipo. Se houverem parâmetros, o ; (ponto-e-vírgula) será utilizado para delimitar a área dos parâmetros. Portanto, um exemplo de Media Type seria text/xml;charset="utf-8” (para descrever um XML cuja codificação seja UTF-8). (SAUDATE, 2014, p. 21)

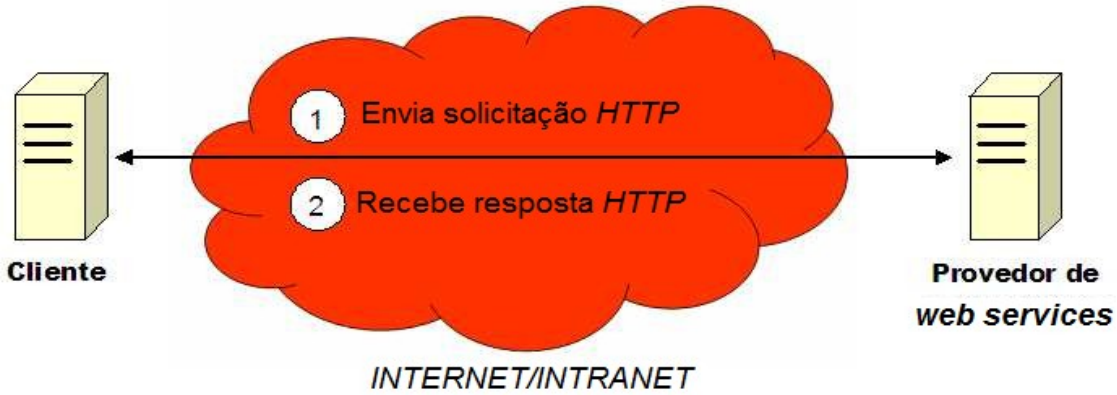
[2] Arrays são objetos semelhantes a listas [...] (MDN, 2017, texto digital)

interligações entre vários recursos diferentes, assim como a utilização adequada dos *headers HTTP*.

Utilizando os quatro verbos do *HTTP*, pode ser obtido um *RESTful*, que por sua vez disponibiliza um grupo de funcionalidades, que são: *Create, Read, Update* e *Delete*, popularmente conhecidos pela sigla *CRUD*, representados pelos respectivos métodos *POST, GET, PUT* e *DELETE*, e podem ser aplicadas a recursos de um sistema (MITCHELL, 2013).

Segundo Gomes e Jandl (2009), a comunicação *REST* é baseada em uma arquitetura simples, onde três componentes são comumente empregados, sendo eles cliente, provedor e o protocolo *HTTP*, representados na Figura 16.

Figura 16 – Representação da comunicação feita através de *Web Service REST*



Fonte: Do autor, adaptado de Gomes e Jandl (2009, p. 98).

Mitchell (2013) afirma que os serviços *RESTful* trabalham com a transferência de representações de estados ou de recursos, onde a representação pode ser feita através de *JSON* ou *XML* ou qualquer outra coisa. Um recurso pode ser um registro individual de um sistema. Pensando em um sistema de *blogs*, os recursos poderiam ser os *posts*, categorias e autores, e para cada um desses recursos existe um *Uniform Resource Identifier (URI)*, que pode ser definido como um caminho para determinado conteúdo, dando nome a este, como pode ser visto na Quadro 1.

Quadro 1 – Exemplos de requisições utilizando *URI's*

Método	**URI**	**Descrição**
GET	*http://localhost/contatos*	Retorna uma lista com todos os contatos.
GET	*http://localhost/contato/10*	Retorna o contato de identificador 10.
GET	*http://localhost/contatos/Lucas*	Retorna todos os contatos que possuem o nome Lucas.
POST	*http://localhost/contato*	Insere um novo contato, os dados devem ser passados no corpo da requisição.
PUT	*http://localhost/contato/3*	Atualiza o contato de identificador 3, os dados devem ser passados no corpo da requisição.
DELETE	*http://localhost/contato/5*	Apaga o contato de identificador 5.

Fonte: Do autor, adaptado de https://www.devmedia.com.br/rest-tutorial/28912

3 FERRAMENTAS RELACIONADAS

Em relação ao acesso ao banco de dados é possível identificar diversas ferramentas do tipo *front-end* que são específicas para cada SGBD, fornecendo acesso a somente os seus bancos de dados. Dentre elas está o pgAdmin, que segundo a documentação, é uma plataforma de administração de banco de dados PostgreSQL, de código livre, que está na versão 4.14, recebendo atualizações frequentes (PGADMIN, 2019).

O phpMyAdmin, assim como o pgAdmin, fornece recursos para a administração de banco de dados, porém essa ferramenta, que é desenvolvida em *PHP* oferece operações compatíveis com os SGBDs MySQL e MariaDB, atualmente esse projeto está na versão 4.9 e desde a seu lançamento em 1998 recebe atualizações (PHPMYADMIN, 2019).

Em ambas as ferramentas citadas anteriormente, é possível a realização de instruções de *DDL* e *DML* tanto pela interface, através de formulários e listagens, quanto escrevendo as instruções em um campo descritivo, nesse caso a realização do *input* da *query* é auxiliada por um *autocomplete.*

Ainda na categoria de ferramentas que fornecem uma camada de interface para a administração de banco de dados, disponibilizando recursos para acesso e manipulação dos dados restritos a apenas um SGBDs, é possível citar mais algumas, dentre as diversas existentes, como por exemplo o Oracle SQL Developer, SQL Server Management Studio, MongoDB Compass e o SQLite Studio.

Além dos aplicativos voltados especificamente para um único SGBD, existem algumas ferramentas que realizam essa interface com diversos SGBDs, A ferramenta DBeaver é um exemplo com essa característica, ela está na versão 6.2 conectando-se a mais

de 80 bancos de dados, oferece muitos recursos tanto para a administração quanto para o monitoramento de banco de dados. Essa ferramenta possui dos tipos de distribuição, sendo uma comunitária e outra *enterprise*, sendo que na distribuição comunitária não existe o suporte para banco de dados NoSQL/BigData (DBEAVER, 2019).

Na mesma categoria do DBeaver, ferramentas que suportam mais de um tipo de banco de dados, está o DataGrip, que também conecta-se a diversos SGBDs. Ele é mantido pela JetBrains, e tem dentre as suas principais funcionalidades está o console de consulta inteligente, que autocompleta o comando e auxilia na refatoração dos códigos SQL (DATAGRIP, 2019).

Além do DBeaver e do DataGrip, existem outras aplicações que fazem parte da categoria das aplicações que fornecem acesso a diversos SGBDs, dentre as diversas existentes é possível citar mais algumas, como por exemplo o Navicat Premium, HeidiSQL, DbVisualizer e o OmniDB.

As soluções citadas anteriormente conectam-se com BDs, fornecendo uma alternativa para a administração e manipulação dos dados. A aproximando-se ainda mais da ferramenta desenvolvida no presente trabalho, serão apresentadas a seguir algumas ferramentas que conectam-se aos bancos de dados através da disponibilização de *APIs*.

Como mencionado anteriormente, já existem algumas aplicações que fornecem a criação de *endpoints*, realizando a conexão entre banco de dados e outras aplicações. Um exemplo é a ferramenta *open source* Fusio, que realiza a disponibilização de uma *API REST* com base em banco de dados relacionais. Nela é possível criar a conexão com as bases de dados e indicar manualmente as rotas desejadas, uma a uma, indicando o nome da tabela e a suas colunas (FUSIO, 2019).

Outra ferramenta que realiza a disponibilização *APIs*, é o Hasura, que possui código aberto e é capaz de conectar-se com banco de dados PostgreSQL e fornecer instantaneamente uma API GraphQL (HASURA, 2019). Das ferramentas que existem, ainda pode-se destacar a Apisentris, que fornece uma *API REST*, conectando-se com os bancos de dados PostgreSQL, MongoDB, MySQL e Google BigQuery, onde são disponibilizados *endpoints* para as bases de dados cadastradas (APISENTRIS, 2019).

Assim como a solução desenvolvida, algumas ferramentas permitem realizar o processo de *DDL* e *DML*. Outras permitem consumir os serviços via *API*. A solução implementada reúne essas características em uma só ferramenta, ainda permitindo conexão com bancos *SQL* e *NoSQL*.

4 METODOLOGIA

Uma das fases de uma pesquisa diz respeito à definição da metodologia para a realização deste presente trabalho. Neste capítulo será apresentado o delineamento da metodologia, detalhando os métodos científicos que foram utilizados.

4.1 Delineamento

O trabalho teve como objetivo desenvolver uma solução com a capacidade de manipular tanto banco de dados relacionais, quanto não relacionais orientados a documentos. Para tal, foram realizadas pesquisas sobre os temas relacionados. Com a exploração desses temas foi possível construir a ferramenta em conformidade com os padrões de software existentes, definindo atributos para que este trabalho seja definido como pesquisa exploratória (SANTOS, 1999).

Outro objetivo do presente trabalho foi realizar uma pesquisa sobre os assuntos que oferecem a base para a sua realização, assim como a construção de uma solução para o problema exposto, avaliando a mesma de forma qualitativa. Como isso o método qualitativo foi abordado nesse estudo, que é determinado por Wainer (2007) como os métodos que possuem fatores que não possibilitam a retirada de medidas, sendo que as variáveis podem apenas serem observadas e discutidas.

Para que a ferramenta fosse construída de acordo com padrões e conceitos já consolidados atualmente, foi necessária a realização de pesquisas bibliográficas, através da análise de textos disponibilizados em livros ou na internet, podendo assim obter um conhecimento antes da construção da solução (FONSECA, 2002).

Após o desenvolvimento da ferramenta, foi realizada avaliação por programadores da área de Engenharia da Computação, Sistemas de Informação e Engenharia de Software. Um questionário qualitativo foi realizado com a intenção de medir e computar a percepção dos recursos e funcionalidades, e ainda verificar se os objetivos da ferramenta foram todos cumpridos. O resultado do questionário foi utilizado para a avaliação da ferramenta construída e os devidos apontamentos e conclusões foram expostos após a análise desse resultado.

4.2 Tecnologias utilizadas

O desenvolvimento da solução contou com a utilização de algumas tecnologias e ferramentas com o objetivo de auxiliar na construção dos processos assim como na execução dos testes e serão detalhadas a seguir.

PHP Hypertext Preprocessor (PHP) é uma linguagem programação interpretada, que dispensa compilação para ser executada no servidor, além disso, ela é uma ferramenta de código livre, focada para o desenvolvimento *web* (PHP, 2018), foi utilizada como a principal linguagem de desenvolvimento da solução. Ainda para a base da ferramenta foi utilizado *o* Adianti *framework 4*, que é um *framework* de código livre focado na criação de sistemas de negócios, servindo para acelerar o desenvolvimento de soluções em *PHP*, oferecendo componentes, recursos e *templates* prontos para a utilização, permitindo que os desenvolvedores possam focar nas regras de negócios da solução, sem ter que se preocupar com os detalhes de programação (ADIANTI, 2018).

Para armazenar os dados da parte gerencial da aplicação foi utilizado a ferramenta PostgreSQL que é um SGBD relacional, de código livre. Essa ferramenta tem como objetivo auxiliar os tanto os administradores de BD a proteger os seus dados assim como gerenciar as suas bases de dados, quanto os desenvolvedores de software na criação de aplicativos. O banco PostgreSQL apresenta diversos recursos como, escalabilidade, extensibilidade, suporte a vários tipos de dados, implementando recursos como integridade dos dados, concorrência, segurança, desempenho e sendo compatível com a Atomicidade, Consistência, Isolamento e Durabilidade (ACID) (POSTGRESQL, 2019).

Pensando no grande volume de dados que os *logs* das requisições da ferramenta criada pode gerar, o armazenamento foi projetado em separado da aplicação. Para essa situação, foi

utilizado o SGBD MongoDB que por sua vez, é arquitetado para facilitar o desenvolvimento e dimensionamento de banco de dados orientados a documentos, sendo altamente escalável. O MongoDB oferece uma versão comunitária, onde cada registro é parecido com a estrutura de um *JSON*, composto por campos e valores. Esse banco de dados, oferece operações do tipo *CRUD*, como visto anteriormente, são as operações que permitem inserir, editar, excluir e recuperar os dados das bases de dados já existentes (MONGO, 2019).

Para a confecção do *front-end* da ferramenta desenvolvida foram utilizadas algumas tecnologias, como por exemplo o JavaScript, que é *c*onhecida como a linguagem de *script* da web. O JavaScript é uma linguagem leve e orientada a objetos, sem distinção entre os tipos e objetos, com tipagem de variáveis dinâmicas das quais não necessitam declaração. O JavaScript é uma linguagem interpretada e por tanto para a execução dos códigos escritos com ele não se faz necessário a compilação (JAVASCRIPT, 2019). Ainda, para a estilização das páginas foi utilizado o Materialize que é um *framework* para *front-ends* responsivos, baseado no Material Design, ele é focado na experiência dos usuários, pois oferece um maior *feedbacks* para os usuários. Com as animações e transições já implementadas, faz com que o trabalho dos desenvolvedores seja mais simples e rápido, e com uma baixa curva de aprendizagem visto que a ferramenta possui uma extensa documentação (MATERIALIZE, 2019).

Como a ferramenta desenvolvida teve como objetivo oferecer um módulo de testes das *APIs* disponibilizadas, o gerenciador de contêineres Docker foi uma opção encontrada para executar tais testes, uma vez que através dessa ferramenta é possível desenvolver utilizando qualquer linguagem sem a preocupação com conflitos entre aplicativos. As linguagens e suas configurações são compactadas em *containers* isolados, podendo ser utilizadas sob demanda, onde os *containers* são criados para executar os testes e removidos logo após, oferecendo uma segurança, impossibilitando a execução de códigos maliciosos (DOCKER, 2019).

5 ESPECIFICAÇÕES DO PROJETO

Neste presente capítulo será abordado a arquitetura da ferramenta, assim como os seus requisitos funcionais e não funcionais. Também nesse capítulo que serão exibidas as interfaces da ferramenta e suas funcionalidades.

5.1 Arquitetura do projeto

Como é mostrado na Figura 17, a arquitetura da solução se divide em quatro principais camadas, que são armazenamento, *front-end*, gerenciador e *API*, detalhados a seguir. A ferramenta construída tem interações realizadas por três tipos de atores, que são os desenvolvedores, os consumidores e os administradores.

I. **Armazenamento:** Abrangendo as bases de dados que são conectadas pela ferramenta, sejam elas do tipo relacional utilizando o SGBD PostgreSQL ou não relacional utilizando o SGBD MongoBD. A ferramenta não possui uma limitação por quantidade, a solução conecta-se a todas as bases de dados que o usuário cadastrar.

II. ***Front-end**:* O *front-end* da solução faz o intermédio entre os desenvolvedores e as bases de dados, é através dele que serão cadastradas as conexões e realizados os processos específicos, através comandos executados manualmente pela interface, tais como as instruções de *DDL* e *DML*. É através dele que os testes da *API* podem ser executados.

III. ***API*:** A camada da *API*, é responsável por manter os *web services* para acesso dos consumidores, é ela que interage com as bases cadastradas através de chamadas *REST* que serão disponibilizadas individualmente para cada tabela ou documento, dependendo do SGBD de cada base de dados. Essa camada também é responsável pela geração de *logs* referentes a todas as requisições realizadas para ela, armazenando assim as informações necessárias para a geração de estatísticas sobre a sua utilização.

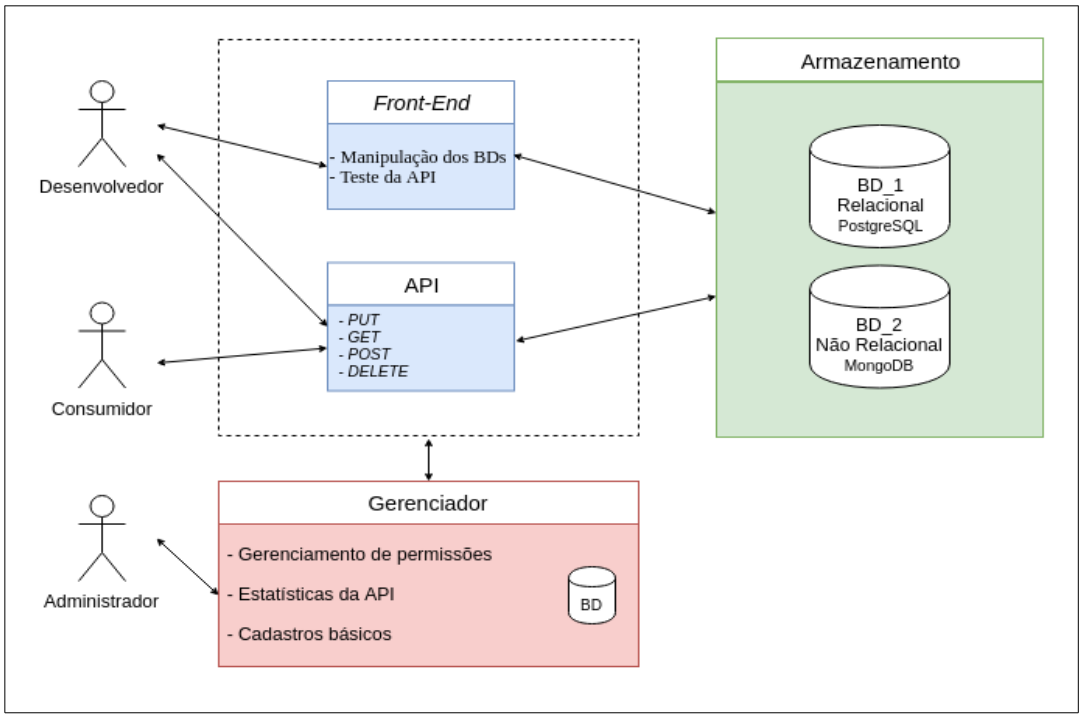
IV. **Gerenciador:** O gerenciador, é a parte da ferramenta que agrupa as informações das bases de dados e das requisições. É também nessa camada que é possível manter os cadastros básicos existentes, assim como gerenciar as permissões de acesso da solução. Ainda é através dessa camada que é possível visualizar as estatísticas da *API*.

O fluxo da solução, como mostrado anteriormente, é baseado na interação de três atores com as quatro camadas da ferramenta. O fluxo inicia-se com o administrador, que é o responsável por manter os cadastros das bases de dados, dos conectores e das permissões do sistema, essa parte pode ser executada através do gerenciador.

Com os devidos cadastros realizados pelo administrador, os desenvolvedores podem acessar as bases de dados através do *front-end* podendo realizar comandos específicos a fim de explorar as bases de dados que estão na camada de armazenamento.

O consumo dos *web services REST*, servem tanto para desenvolvedores quanto para consumidores, onde ambos podem realizar requisições para a camada da *API*, essa por sua vez interage tanto com a camada de armazenamento, para execução da instrução solicitada na base de dados, quanto com a camada do gerenciador, para a verificação de permissões e também para obter as informações necessárias para estabelecer a conexão com a base de dados na qual será executado o comando, e só após a execução dessas duas partes será retornado o resultado fornecido pela camada de armazenamento.

Figura 17 – Representação da arquitetura da ferramenta



Fonte: Do autor, 2019.

5.2 Requisitos da ferramenta

Para o desenvolvimento do projeto, alguns requisitos funcionais foram implementados, assim como, alguns requisitos não funcionais também foram definidos para que o funcionamento da solução estivesse adequado.

A seguir, no Quadro 2, são listados os requisitos funcionais, onde além da descrição de cada um deles serão especificados a sua prioridade e a camada da aplicação a qual ele pertence.

Quadro 2 – Requisitos funcionais

Descrição do requisito	**Prioridade**	**Camada**
1 – Conectar com banco de dados relacional	Alta	Armazenamento
Permitir que a ferramenta cominuque-se com bancos de dados relacionais, utilizando o SGBD PostgreSQL.		
2 – Conectar com banco de dados não relacional	Alta	Armazenamento
Permitir que a solução interaja com bancos de dados não relacionais da categoria de documentos, utilizando o SGBD MongoDB.		

Descrição do requisito	Prioridade	Camada
3 – Permitir utilização de várias bases de dados	Alta	Armazenamento
Permitir que sejam adicionadas múltiplas bases de dados de ambos os SGBDS, mostrados nos requisitos funcionais 1 e 2.		
4 – Disponibilizar *API REST*	Alta	*API*
Disponibilizar *API REST*, implementando os métodos *GET*, *PUT*, *DELETE* e *POST*, tanto para as tabelas dos bancos de dados relacionais quanto para as coleções dos bancos de dados orientado a documentos.		
5 – Permitir definir permissões por estrutura na *API*	Alta	*API*
Permitir que sejam concedidos permissões de leitura e escrita para cada uma das estruturas dos conectores cadastrados, seja elas tabelas ou coleções.		
6 – Gravar dados da *API REST*	Alta	*API*
Ao receber uma requisição para a *API REST*, a mesma deve armazenar as informações desse *web service* em banco de dados, independentemente do método requisitado.		
7 – Permitir definir grupos de permissões para usuários	Alta	Gerenciador
Manter cadastro de grupos com permissões específicas para acesso aos bancos de dados, como escrita e leitura.		
8 – Permitir visualização de estatísticas de utilização da *API*	Alta	Gerenciador
Disponibilizar um painel de estatísticas por base de dados, contendo a informações obtidas através da utilização da *API REST*.		
9 – Permitir manipulação das bases de dados	Alta	*Front-end*
Permitir que os usuários utilizem o sistema para manipular as bases de dados previamente cadastradas, possibilitando execução de comandos específicos e pontuais		
10 – Permitir visualização das bases de dados	Alta	*Front-end*
Permitir visualizar estrutura das bases de dados já cadastradas, como colunas para bancos relacionais e atributos para.		

Fonte: Do autor, 2019.

Assim como os requisitos funcionais, para a construção da ferramenta alguns requisitos não funcionais também foram atendidos, dos quais estão listados no Quadro 3. Para

os requisitos não funcionais não existe a distinção de camadas, já que as mesmas ferramentas foram utilizadas para a construção de toda a ferramenta.

Quadro 3 – Requisitos não funcionais

Descrição do requisito	**Prioridade**
1 – Utilizar *PHP* 7.2	Alta
Utilizar linguagem de programação *PHP* na versão 7.2 para a construção do *back-end* da ferramenta.	
2 – Utilizar *framework* Adianti	Alta
Utilizar o *framework* Adianti para auxiliar e otimizar a construção do *back-end* da ferramenta.	
3 – Ser *WEB*	Alta
A ferramenta deverá ser *web*, podendo ser acessado de diversos locais, não sendo necessário a responsividade para acesso *mobile.*	
4 – Ser compatível com o navegador Google Chrome	Alta
Manter compatibilidade da ferramenta com o navegador Google Chrome na versão 70 ou superior.	
5 – Utilizar *framework* Materialize para o desenvolvimento das interfaces	Alta
Utilizar o *framework* Materialize para auxiliar e otimizar o desenvolvimento da interface da solução.	
6 – Utilizar autenticação para a *API*	Alta
Utilizar autenticação de acesso básico para todas as requisições de *web services* realizadas para a *API* através de um *key* que deverá ser passada no cabeçalho de todas as requisições.	
***7* – Conectar com o banco de dados PostgreSQL**	Alta
Permitir conexão com o SGBD relacional PostgreSQL na versão 11 ou superior, para disponibilização na *API REST*.	
8 - Conectar com o banco de dados MongoDB	Alta
Permitir a conexão com o SGBD não relacional e orientado a documentos, MongoDB, na versão 4 ou superior, para disponibilização na *API REST*.	
9 – Utilizar banco de dados PostgreSQL	Alta
Utilizar o SGBD relacional PostgreSQL, na versão 11 ou superior, para armazenar os dados que serão utilizados pela ferramenta.	

Descrição do requisito	Prioridade
10 – Utilizar banco de dados MongoDB	Alta
Utilizar o SGBD não relacional e orientado a documentos, MongoDB, na versão 4 ou superior, para armazenar as informações das requisições feitas para a *API REST*	

Fonte: Do autor, 2019.

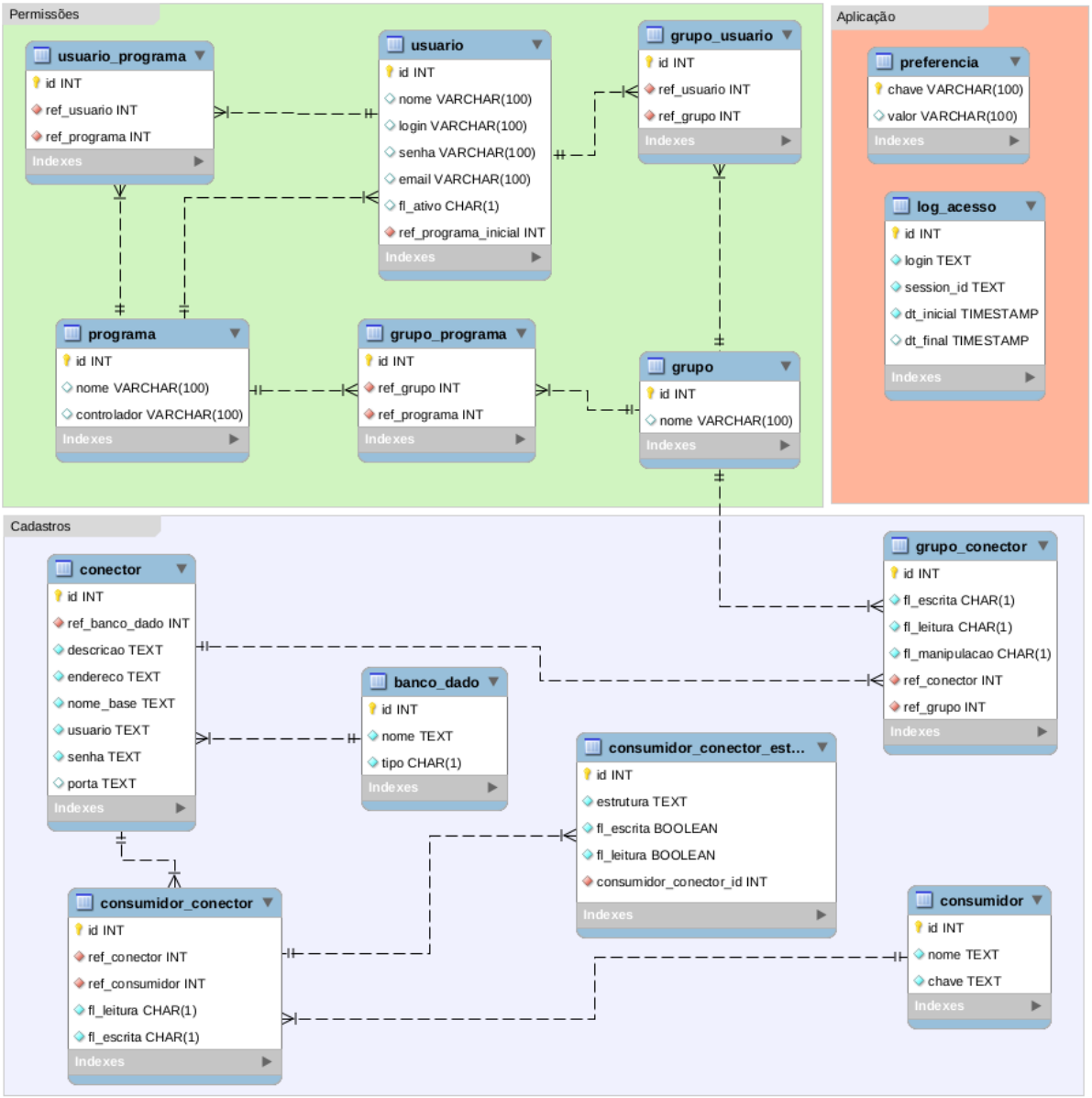
5.3 Bases de dados da aplicação

Para que os requisitos mencionados anteriormente pudessem ser atendidos, foi elaborado um modelo de Entidade-Relacionamento (ER) e um esboço de documento, que serviu como referência para a criação das bases de dados que a ferramenta utiliza para armazenar as informações necessárias para a sua concepção. Imaginando que a quantidade de requisições que a *API* será grande, projetou-se um isolamento em duas partes para armazenamento das informações da solução. Uma parte, responsável pelo armazenamento das informações utilizadas no *front-end* e no gerenciador, ou seja, na parte gerencial da solução, e outra parte responsável pelo armazenamento dos *logs* da *API*.

O modelo ER, foi utilizado como base para a camada gerencial da ferramenta, como é mostrado na Figura 18, foi dividido em três grandes partes. A parte de permissões é responsável por manter as informações de usuários, grupos e programas, controlando assim, os acessos ao sistema. O segundo grupo de tabelas do diagrama é a nominada de aplicação, e como o próprio nome diz, ela possui a responsabilidade de manter as preferências e os *logs* de acesso da aplicação. Por último, a parte dos cadastros, que é responsabilizada pelo armazenamento dos dados dos conectores, consumidores e dos bancos de dados.

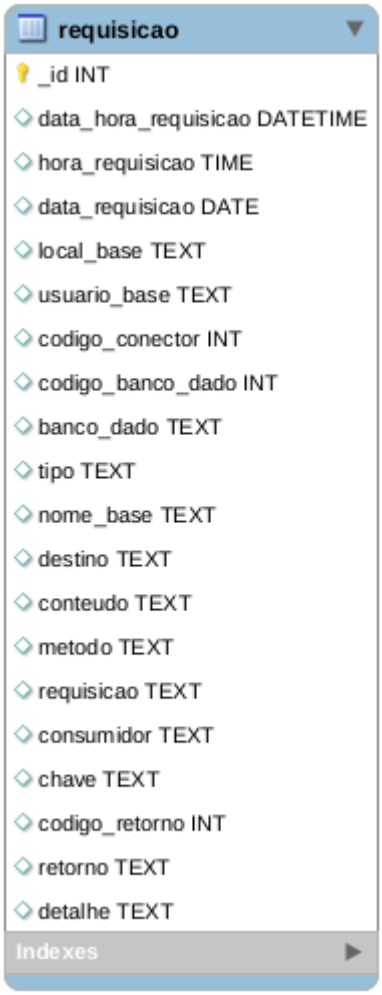
Para segunda parte do armazenamento, que possui a responsabilidade de guardar os dados de todas as requisições feitas para *API*, foi utilizado o banco de dados MongoDB, e para tal, foi elaborado um modelo de documento contendo as informações dos conectores, consumidores e da própria requisição, como é mostrado na Figura 19.

Figura 18 – Modelo Entidade-Relacionamento da ferramenta



Fonte: Do autor, 2019.

Figura 19 – Modelo de documento para a gravação de *logs* da *API*



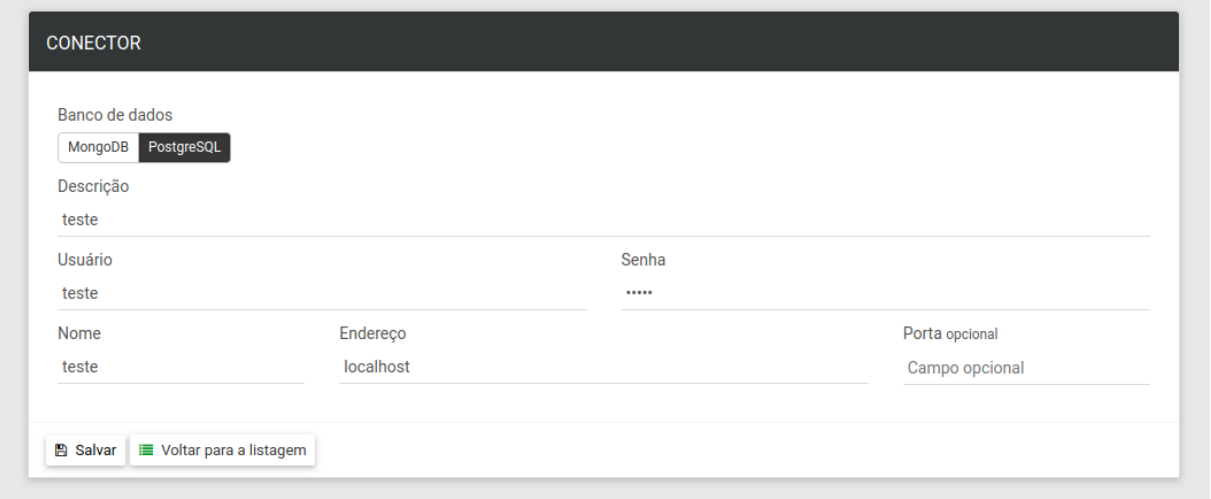
Fonte: Do autor, 2019.

5.4 Interfaces e funcionalidades da ferramenta desenvolvida

Algumas imagens da interface e funcionalidades da ferramenta que foi desenvolvida, serão apresentados nesta seção.

A Figura 20 mostra a interface de cadastro de conectores, no qual pode ser efetuado a conexão com uma base de dados. Nessa página pode-se escolher o tipo do BD, sendo possível o tipo PostgreSQL ou MongoDB, assim como um nome para o conector. Nesta mesma tela é realizada a configuração de acesso à base de dados, através dos dados de endereço, porta, nome, usuário e senha.

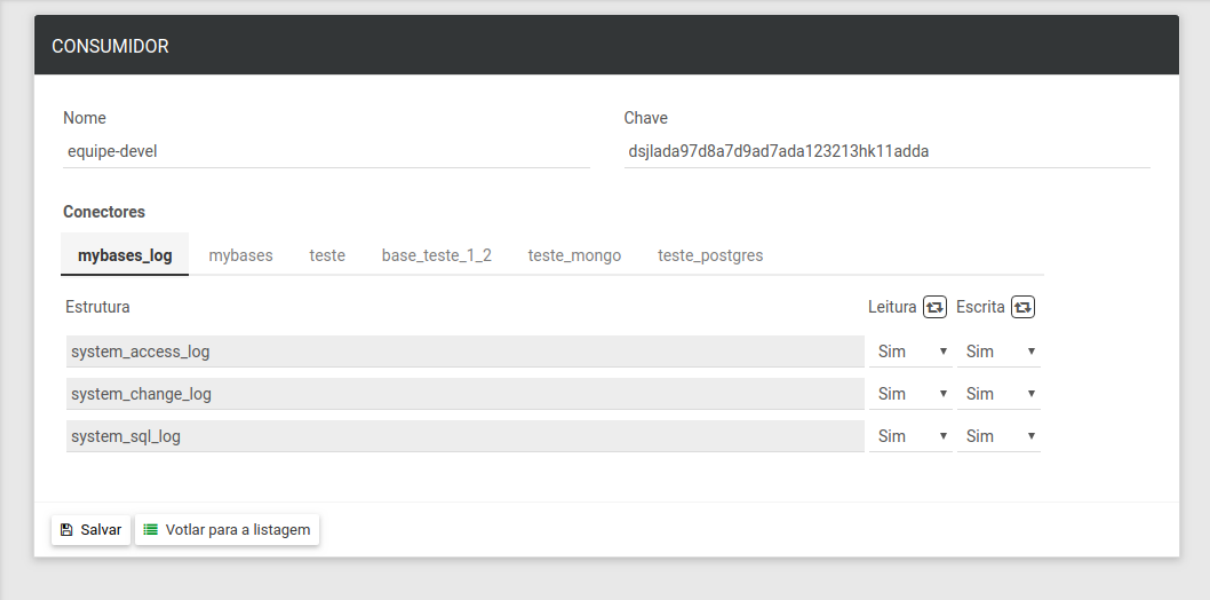
Figura 20 – Interface para cadastro de um conector com uma base de dados



Fonte: Do autor, 2019.

Para cadastrar os consumidores é necessário informar um nome e uma chave, ambos campos utilizados para a autenticação e identificação no momento que é realizado uma requisição para a *API*. Também nesse momento, ocorre a criação de um consumidor que define as liberações de leitura e escrita para as estruturas dos conectores. Como pode ser visto na Figura 21, são exibidas todas os conectores e suas estruturas.

Figura 21 – Interface para cadastro de um consumidor

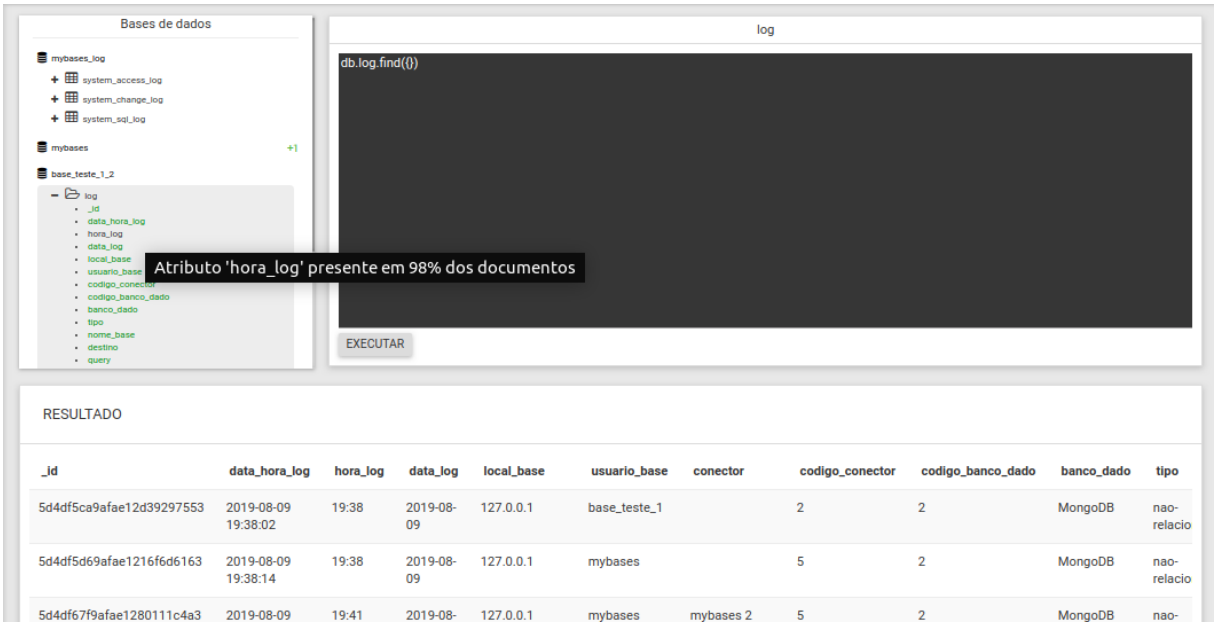


Fonte: Do autor, 2019.

Na Figura 22 é apresentado a página da ferramenta em que os desenvolvedores podem executar alguma instrução específica, seja para testes, para ajustes, ou para qualquer outra finalidade que for necessária realizar diretamente na base. Essa tela é um interpretador de

comandos, onde as instruções devem ser escritas de acordo a linguagem aceita pelo tipo do banco de dados do conector selecionado. Antes da execução de cada comando na base de dados é verificado se o usuário logado possui a permissão para concluir o comando, seja ela de leitura, escrita ou de manipulação. Nessa tela são apresentados todos os conectores, cujo o usuário tenha liberação, expandindo os conectores serão apresentadas as estruturas que o mesmo contém, sejam elas tabelas ou coleções. Ao abrir uma estrutura será exibido a lista dos seus atributos ou colunas, e caso o conector for do tipo MongoDB, será mostrado a porcentagem de ocorrência do atributo nos documentos da coleção selecionada.

Figura 22 – Interface do explorador de base de dados



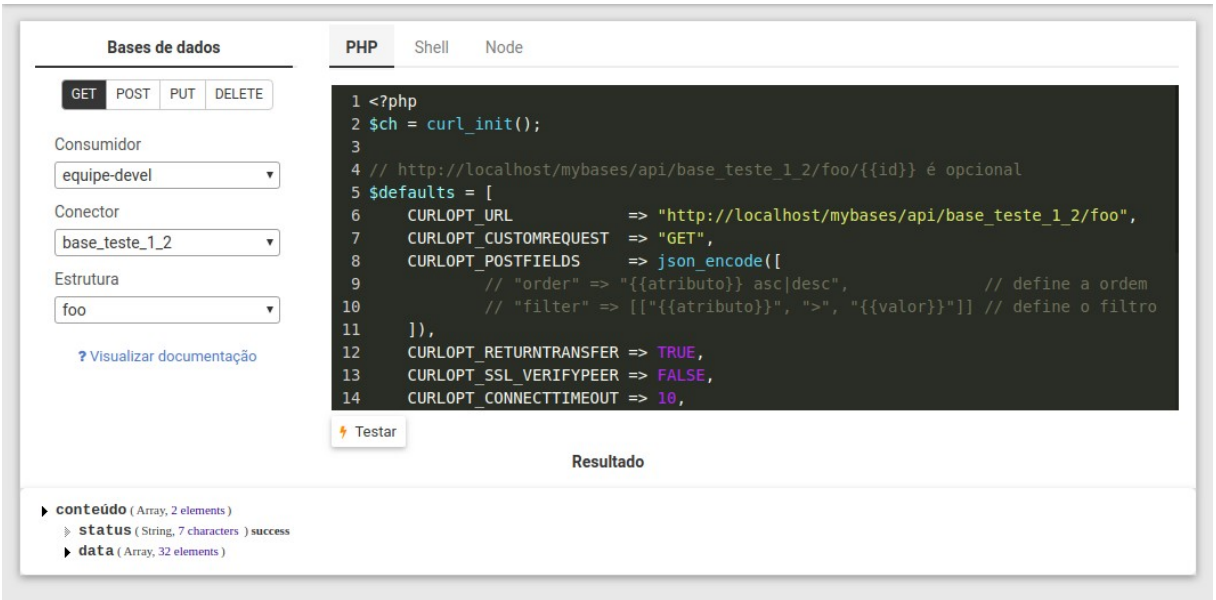
Fonte: Do autor, 2019

Após efetuar os cadastros dos conectores e consumidores, os usuários da ferramenta podem verificar o funcionamento da *API REST*, através do módulo de Teste da *API*. Esse módulo permite a execução de requisições, para os quatro métodos implementados pela solução, que são *GET*, *POST*, *PUT* e *DELETE*. As requisições podem ser efetuadas utilizando três tecnologias distintas, sendo elas *PHP*, NodeJS e *Shell Script*. Esses testes simulam o ambiente real de utilização, executando as instruções no conector selecionado, verificando as permissões de acordo com o consumidor selecionado. Todo o retorno recebido da execução desses testes são exibidos abaixo do editor.

Os trechos de códigos fornecidos como exemplo para cada uma das tecnologias, são funcionais, e os mesmos podem ser utilizados fora do ambiente de testes. Esses exemplos de códigos podem facilmente ser alterados, deixando assim, os testes dinâmicos e sob controle

do usuário. Visando a segurança e estabilidade da ferramenta, é criado contêiner Docker para cada teste realizado, gerando assim uma camada de isolamento, onde os testes são executados em um ambiente à parte, que é excluído logo após o seu término. A interface do módulo Testes da *API* pode ser visualizada na Figura 23.

Figura 23 – Interface do módulo de Testes da *API*



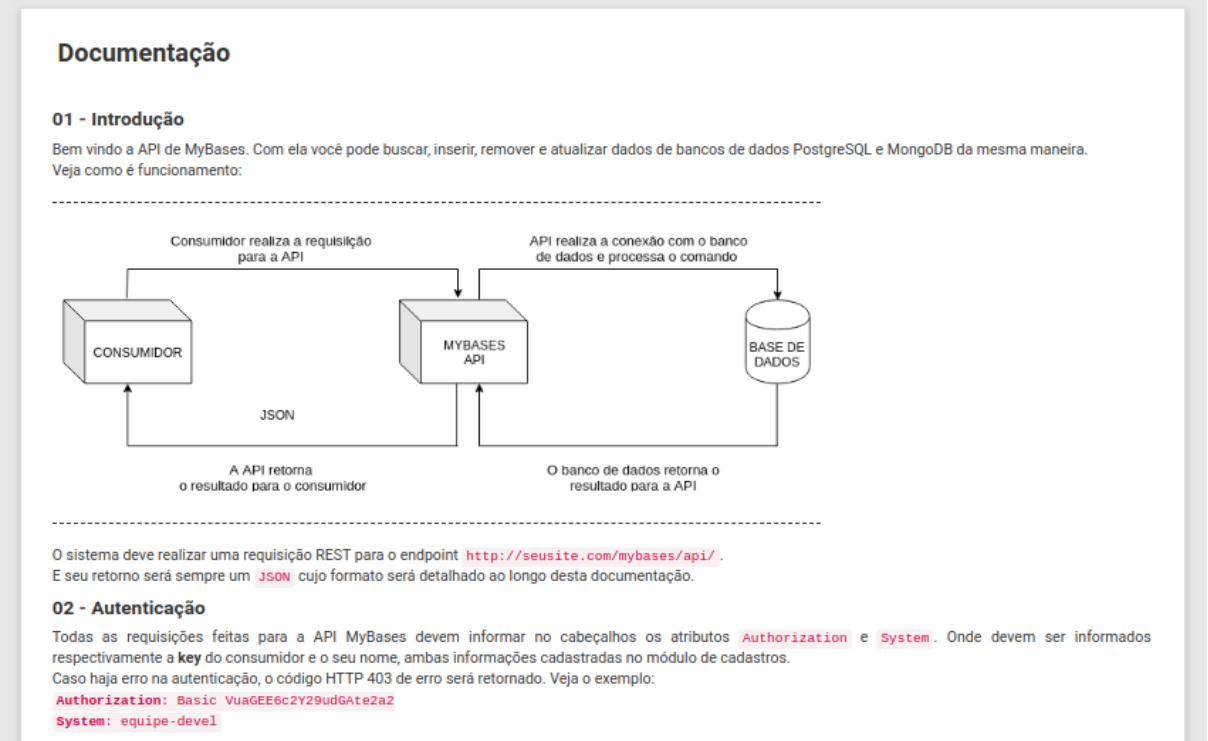
Fonte: Do autor, 2019.

Para que a utilização da *API* fosse completamente entendida, e para que os usuários pudessem conhecer todas as funcionalidades disponíveis, seja em relação a inserção, edição, exclusão ou recuperação de registros, foi disponibilizado uma página de documentação, focada no funcionamento da *API*. Como pode ser visto na Figura 24, a página de documentação foi construída utilizando *Hypertext Markup Language (HTML)*, e disponibilizada dentro da própria ferramenta, podendo assim ser facilmente acessada durante a execução dos testes.

A página de documentação é dividida em quatro tópicos, sendo eles: 01 – introdução, que fornece um panorama do funcionamento da *API*; 02 – autenticação, que mostra como preencher os dados do consumidor no cabeçalho da requisição; 03 – retorno, que explica o formato que o resultado das requisições são devolvidos; e, 04 – serviços, esse por sua vez, é subdividido em quatro partes, cada uma dedicada a explicar um tipo de serviço, sendo eles inserção, remoção, atualização e buscas, esta última separada em dois tipos: busca simples, que retorna apenas um registro e busca de múltiplos registros, retornando os registros de

acordo com os parâmetros repassados na requisição, que podem ser filtros, ordenações, limites e deslocamentos, todos esses detalhados na documentação.

Figura 24 – Documentação da *API*



Fonte: Do autor, 2019.

Na Figura 25 é mostrado a interface para a visualização das estatísticas dos bancos de dados, onde o usuário poderá escolher filtros de datas assim como selecionar o conector do banco de dados do qual serão recuperadas as informações para gerar as estatísticas. A ferramenta disponibiliza quatro gráficos para a visualização de estatísticas, sendo eles:

a) Utilização por estrutura: mostra o nome da estrutura e a quantidade de requisições realizadas para ela, independentemente do tipo da requisição e da situação.

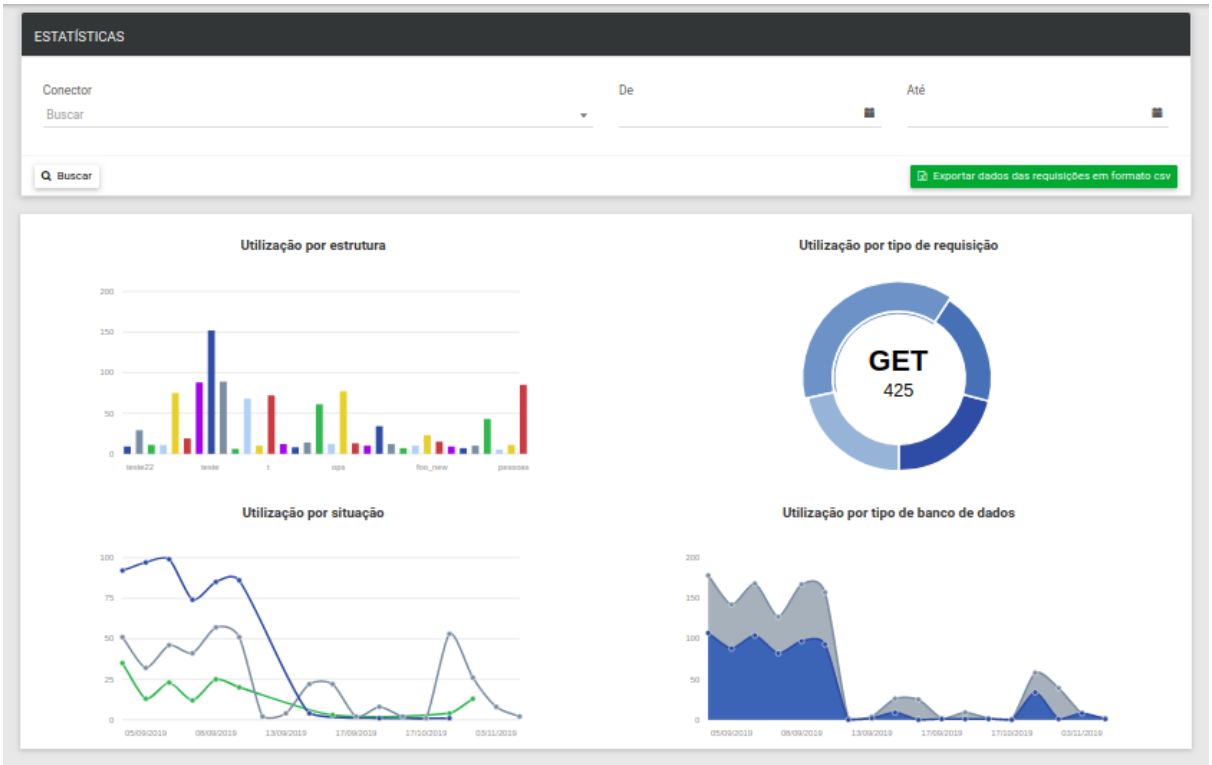
b) Utilização por tipo de requisição: mostra a quantidade de requisições realizadas agrupadas pelo seu tipo, sendo eles *GET*, *POST*, *DELETE*, *UPDATE*. Esse gráfico não filtra as situações.

c) Utilização por situação: mostra a quantidade de requisições agrupadas por situação, podendo ser: *success*, que são requisições que obtiveram sucesso; *client error*, que as requisições em que ocorreram erros na chamada do *web service* por parte do usuário; e, *server error*, que são erros no processamento da requisição ocorridos no servidor.

d) Utilização por tipo de banco de dados: mostra a quantidade de requisições agrupadas pelo tipo de banco de dados.

Além dos gráficos, ainda é fornecido uma opção de *download* de todas as informações das requisições em uma planilha no formato *Comma Separated Values (CSV)*, deixando a cargo do administrador, agrupar as informações da maneira desejada.

Figura 25 – Interface da página de estatísticas



Fonte: Do autor, 2019.

5.5 Processamento da *API*

As conexões processadas pela *API*, foram restringidas a dois tipos de banco de dados PostgreSQL e MongoDB, como já mencionado anteriormente. Esses dois bancos foram escolhidos por serem diferentes tanto na estrutura de armazenamento quanto no acesso aos dados armazenados neles, demostrando assim, que é possível, através de um *middleware*, fornecer o acesso e a manipulação de dados de maneira transparente aos desenvolvedores.

Mesmo tendo selecionado apenas dois tipos de banco de dados, a ferramenta foi arquitetada para que outros bancos de dados sejam facilmente incorporados na sua estrutura. Considerando que os SGBDs possuem formas específicas de comunicação, a adição de mais

um tipo de banco de dados é possível, porém, necessita da programação da classe[3] que faz a conexão.

Como a ferramenta foi projetada considerando a possibilidade dessas novas conexões, a implementação pode ser realizada através de dois simples passos. Sendo o primeiro passo a adição da nova classe de conexão na estrutura da classe da fábrica, representada na Figura 26, que é responsável por retornar a classe que contém os serviços de manipulação do banco de dados. O segundo passo é o desenvolvimento da classe que contém os serviços de manipulação do banco dados, responsável pela ligação entre a chamada da *API* e o banco de dados. Essa classe deve implementar a interface[4] Conexao, representada na Figura 27, seguindo assim um padrão.

Figura 26 – Classe fábrica retorna o serviço do banco de dados

```
class ConexaoFactory
{
    public static function getConexaoService($tipo)
    {
        $conexao = NULL;

        if($tipo == BancoDado::MONGODB)
        {
            $conexao = new ConexaoServiceMongoDB;
        }
        else if($tipo == BancoDado::POSTGRESQL)
        {
            $conexao = new ConexaoServicePostgreSQL;
        }
        else
        {
            throw new Exception("Ainda não implementado");
        }

        return $conexao;
    }
}
```

Fonte: Do autor, 2019.

[3] “As classes de programação são projetos de um objeto, aonde têm características e comportamentos, ou seja, permite armazenar propriedades e métodos dentro dela. Para construir uma classe é preciso utilizar o pilar da abstração. Uma classe geralmente representa um substantivo, por exemplo: uma pessoa, um lugar, algo que seja abstrato” (PALMEIRA, 2012, texto digital).

[4] “Podemos definir como interface o contrato entre a classe e o mundo exterior. Quando uma classe implementa uma interface, se compromete a fornecer o comportamento publicado por esta interface. As classes ajudam a definir um objeto e seu comportamento e as interfaces que auxiliam na definição dessas classes. As interfaces são formadas pela declaração de um ou mais métodos, os quais obrigatoriamente não possuem corpo.” (WELLINGTON, 2010, texto digital).

Figura 27 – Classe de modelo para conexão entre a *API* e os bancos de dados

```php
interface Conexao
{
    /**
     * Grava dados na estrutura
     * @param  String $estrutura Tabela/Coleção
     * @param  array  $dados     Dados a serem inseridos
     * @return bool              T/F
     */
    public function insert($estrutura, $dados);

    /**
     * Atualiza registro na estrutura
     * @param  String $estrutura      Tabela/Coleção
     * @param  String $identificador  Identificador do registro
     * @return bool                   True or Trhow
     */
    public function update($estrutura, $identificador, $dados);

    /**
     * Busca registro na estrutura
     * @param  String $estrutura      Tabela/Coleção
     * @param  String $identificador  Identificador do registro
     * @return array                  Registro [atributo => valor]
     */
    public function find($estrutura, $identificador);

    /**
     * Busca registros na estrutura
     * @param  String $estrutura     Tabela/Coleção
     * @param  array  $filtros       Filtros      [ ['chave', 'operador', 'valor', 'expressao'] ]
     * @param  array  $propriedades  Propriedades [limit, offset, order]
     * @return array                 Array de registros [[atributo => valor, ... ], ... ]
     */
    public function list($estrutura, $filtros, $propriedades);

    /**
     * Remove um registro na estrutura
     * @param  String $estrutura      Tabela/Coleção
     * @param  String $identificador  Identificador do registro
     * @return bool                   True or Trhow
     */
    public function delete($estrutura, $identificador);
}
```

Fonte: Do autor, 2019.

Com as novas classes de conexões sendo criada seguindo o modelo estabelecido pela classe de interface mostrada na Figura 27, garantimos que toda nova classe de conexão com banco de dados terá o mesmo funcionamento, possuindo a mesma assinatura, mudando apenas o seu nome, sendo assim, ao adicionar novas integrações com outros tipos de banco de dados, como por exemplo SQLite, MySQL, Oracle ou Cassandra garantimos que o seu funcionamento será igual.

6 RESULTADOS E DISCUSSÕES

Com o objetivo de analisar a ferramenta criada nesse trabalho foi elaborado um questionário para coletar o *feedback* avaliativo dos usuários que utilizaram a ferramenta. Conforme Apêndice A, o questionário foi elaborado contendo doze perguntas, sendo elas, nove perguntas objetivas e três descritivas. As questões foram criadas tanto para coletar informações sobre a percepção dos usuários sobre a utilização da ferramenta, quanto para estabelecer um perfil dos entrevistados.

As entrevistas foram realizadas com dez profissionais da área de desenvolvimento de software, sendo envolvidos estudantes de Engenharia de Software, Engenharia da Computação, Sistemas de Informação, assim como engenheiros de Software e de Computação já graduados.

Dentre todos os entrevistados 90% trabalham com o desenvolvimento de software no dia a dia, e 10% deles atuando com o gerenciamento de equipes, indiretamente no desenvolvimento. Todos os usuários que utilizaram a aplicação possuem mais de dois anos de experiência em suas profissões, e apenas 20% deles atuam a mais de seis anos.

O processo de entrevista ocorreu em três etapas distintas, e conduzidas individualmente. A primeira etapa ficou reservada para uma breve apresentação da ferramenta, explicando o objetivo principal da mesma. Após a apresentação foi disponibilizado um roteiro, que em suma, é uma sequência de ações que os entrevistados deveriam realizar, tendo a finalidade de testar as funcionalidades existentes na ferramenta. Como pode ser visto no Apêndice B, o roteiro continha treze passos, que iniciava-se no *login* na ferramenta e terminava com a visualização das estatísticas das interações realizadas

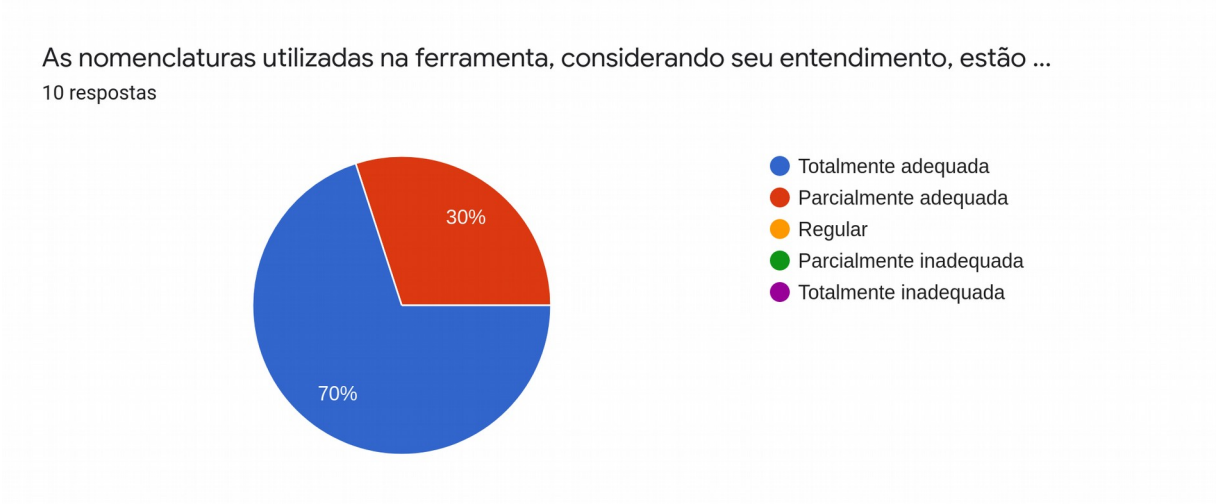
durante o roteiro. A terceira e última etapa foi utilizada para a realizar coleta dos *feedbacks*, através do questionário mencionado anteriormente.

A análise das questões começará a partir da terceira, pois as duas primeiras perguntas do questionário serviram para identificar o perfil dos usuários, quanto ao tempo de atuação na sua profissão, assim como qual é o seu relacionamento com o desenvolvimento de software.

A questão de número três visava obter a informação sobre a usabilidade da ferramenta, e conforme o esperado, as respostas foram positivas e se dividiram em dois grupos, onde 50% definiram como totalmente apropriada e 50% parcialmente apropriada. Percebe-se então, que o entendimento das funcionalidades e a usabilidade da ferramenta está apropriada.

Quanto às nomenclaturas utilizadas na interface e na documentação da solução, os avaliadores declararam estar adequadas, como pode ser visto na Figura 28, as respostas da quarta questão ficaram divididas entre totalmente adequada (70%) e parcialmente adequada (30%). Essa questão era uma das perguntas-chave, uma vez que a definição dos termos é de suma importância, pois em diversos momentos foi preciso referenciar nomenclaturas de componentes existente nos bancos de dados, porém sem especificá-los, já que a ferramenta tem como objetivo suportar diversos tipos. Um exemplo foi a utilização do termo [estrutura], para referenciar as tabelas dos bancos relacionais e as coleções dos bancos não relacionais, assim como a utilização do termo [atributo] para referenciar os campos dos documentos e as colunas das tabelas.

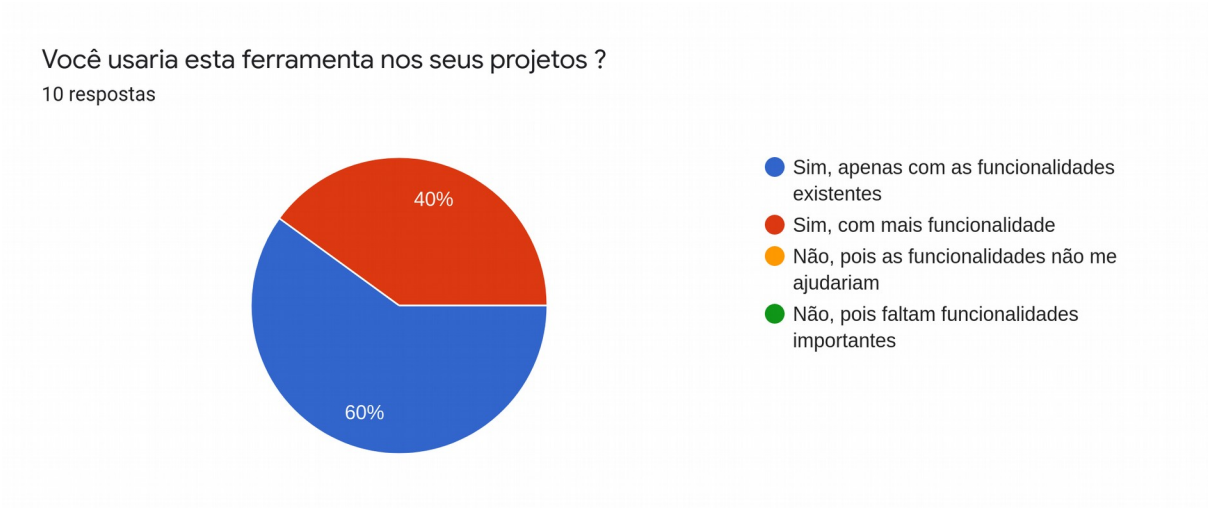
Figura 28 – Respostas da quarta pergunta, referente as nomenclaturas utilizadas na ferramenta



Fonte: Do autor, 2019.

Todos os avaliadores declaram que utilizariam a ferramenta nos seus projetos, sendo que 60% deles usufruiriam sem efetuar nenhuma mudança e outros 40% usariam caso novas funcionalidades fossem adicionadas à ferramenta. Nenhum deles respondeu que a ferramenta não ajudaria ou que não utilizariam devido à falta de fundacionalidades importantes, como pode ser verificado na Figura 29, que mostra os resultados da quinta pergunta.

Figura 29 – Respostas da quinta pergunta, referente a utilização da ferramenta em projetos



Fonte: Do autor, 2019.

Na sexta pergunta, os avaliadores foram questionados sobre as funcionalidades da ferramenta, tendo as seguintes três opções de respostas: Suficientes, atendem a proposta; Suficientes, mas com ressalvas, pois faltam alguns recursos; Insuficientes, ferramenta incompletas. Perante os objetivos explicados antes da realização do roteiro, 100% dos avaliadores indicaram que as funcionalidades são suficientes e atendem a proposta do presente trabalho.

Em relação a contribuição que uma *API* com os recursos que a ferramenta desenvolvida possui, como mostrado anteriormente, todos os avaliadores afirmaram que sim, existe uma contribuição para o desenvolvimento de software, conforme respostas fornecidas na pergunta de número sete. Através da pergunta de número oito, que questionava sobre a dificuldade de trabalhar com diversos tipos de banco de dados, os mesmos 100% perceberam que existe essa dificuldade, e uma ferramenta desse tipo auxiliaria o processo de desenvolvimento.

Sobre a documentação que foi disponibilizada dentro da ferramenta, e que os avaliadores tiveram que utilizar durante a realização do roteiro, todos responderam na última

pergunta quantitativa, que a mesma ficou clara, e que conseguiram identificar como usufruir os recursos que a *API* oferece para buscar registros, funcionalidade essa, com maior possibilidades de configurações.

As últimas perguntas foram destinadas a coletar a opinião dos entrevistados, onde os mesmos foram questionados sobre pontos positivos e negativos da solução, assim como tiveram um espaço para expor os seus comentários, críticas e sugestões.

As respostas da pergunta de número dez, que visava identificar pontos positivos da ferramenta, teve sete respostas, que estão listadas no Quadro 4. Através das respostas é possível destacar a intuitividade e usabilidade que a ferramenta oferece, facilitando a utilização da mesma. Outro ponto a ser destacado, é que o uso da ferramenta torna o processo mais rápido e simples.

Quadro 4 – Respostas sobre pontos positivos da ferramenta

Respostas
Ferramente fácil de usar, bastante intuitiva.
A documentação bem clara facilita o uso da *API*, que também é bem tranquila de usar. O principal ponto positivo seria a facilidade que traz ao consumidor para a comunicação com diversas bases de dados diferentes.
Ficou bastante intuitiva e fácil de utilizar. O front-end é bonito e não é poluído visualmente
Usabilidade e Layout intuitivo da ferramenta Facilidade nos cadastros, Geração de códigos em diversas plataformas
Documentação, dashboard, visualização de diferentes tipos de rest
A existencia dela
Facilidade de utilização
Acredito que a ferramenta é muito útil e resolve o problema das diferentes bases de dados, tornando o processo mais rápido e simples e facilitador para o consumidor.

Fonte: Do autor, 2019.

No Quadro 5, são mostradas as respostas sobre os pontos negativos da ferramenta, que os entrevistados destacaram na pergunta onze. Alguns itens foram observados, dentre eles pode ser destacado a falta de clareza na opção de adicionar uma nova estrutura, que foi representado pelo ícone *+1* (mais 1), que por sua vez foi encontrado com certa dificuldade por alguns dos entrevistados. Outro item mencionado por alguns dos usuários, foi a falta da

existência de uma transação, onde alguns dos respondentes desejavam poder desfazer uma operação realizada. Esse último, por sua vez, não estava estabelecido no escopo do presente trabalho e foi adicionado na lista de trabalhos futuros.

Quadro 5 – Respostas sobre os pontos negativos da ferramenta

Respostas
nenhum
Somente Postgres e Mongo
+1 bem escondido
Não destaco nada como negativo, apenas a aprimorar.
Não possui um controle de transação
Não poder dar rollback

Fonte: Do autor, 2019.

A última questão, foi destinada a disponibilizar um espaço para que os entrevistados pudessem deixar seus comentários, críticas construtivas e sugestões relacionadas com a ferramenta. Todas as respostas foram devidamente anotadas, como pode ser visto no Quadro 6, e algumas delas foram adicionadas na lista de trabalhos futuros.

Quadro 6 – Comentários, críticas construtivas e sugestões relacionadas a ferramenta

Respostas
O roteiro de testes possui alguns itens que estão muito extensos e, mesmo que tenham clareza das tarefas, me perdi e esqueci tudo que precisava ser feito tendo que ler novamente o item.
Portar para MySQL, SQLServer, SQLite, Oracle Ter um gerador de relatórios dinâmico da base de dados do cliente em formatos como xls, txt, cvs e pdf Ter exemplos de utilização direto na tela do explorador de base dados
Atributos que devem ser substituidos no teste da *API* não ficaram claros.

Fonte: Do autor, 2019.

7 CONSIDERAÇÕES FINAIS

Através dos estudos concretizados por esse trabalho, constatou-se que a importância de existir uma ferramenta que auxilie na construção de aplicações oferecendo uma maneira em que o acesso e a manipulação de bases de dados seja padronizada, é verdadeira.

Observando as respostas dos entrevistados, constatou-se a dificuldade de trabalhar com diversos tipos de banco de dados é realidade, e uma ferramenta com as características como a desenvolvida, ajudaria a minimizar a dificuldade desse trabalho.

A utilização de bancos de dados é imprescindível na sociedade atual e sempre utilizado em aplicações, mesmo que de forma simplista. Esse trabalho visa facilitar esse uso, proporcionando uma utilização rápida e simples para profissionais com pouca experiência, e até mesmo sem o conhecimento dos comandos específicos dos bancos de dados suportados pela ferramenta.

Por meio das avaliações realizadas com profissionais da área de desenvolvimento de software, foi possível identificar a satisfação que os mesmos tiveram em relação a ferramenta. Os entrevistados destacaram a facilidade de utilização e rapidez proporcionada pela ferramenta durante o processo de desenvolvimento.

Outro ponto que mostra o sucesso deste trabalho é que todos os profissionais que avaliaram a ferramenta evidenciaram possuir a vontade de utilizar a ferramenta em seus projetos.

Algumas considerações relatadas durante o processo de avaliação da ferramenta foram colocadas na lista de trabalhos futuros, como mencionado anteriormente, onde alguns dos

pontos são relacionados a novas implementações e outros envolvem ajustes nas funcionalidades já existentes.

A inclusão de novos bancos de dados suportados pela ferramenta é uma das principais melhorias a serem feitas, assim como a adição de exemplos de instruções de cada banco de dados no explorador de base de dados a fim de facilitar as operações que são mais específicas. No *roadmap* está outra grande funcionalidade, que é a implementação de um controle de transação, possibilitando os desenvolvedores desfazerem e confirmarem os comandos requisitados para a *API*.

REFERÊNCIAS

ADIANTI. **Adianti Framework.** Disponível em: <http://www.adianti.com.br>. Acesso em: 11 de abr. 2019.

APISENTRIS. **Apisentris**. Disponível em <https://apisentris.com/>. Acesso em: 06 nov. 2019.

BRITO, Edson Gouveia. **Cardinalidade**. Disponível em <http://edygouveia.blogspot.com/2013/04/cardinalidade-sql-server.html>. Acesso em: 15 dez. 2019.

CORREIA, Karina da Silva. **Evolução arquitetural de um web service: transformação de código e avaliação da arquitetura**. 2015. Disponível em: <https://repositorio.ufpe.br/bitstream/123456789/17765/1/Disserta%C3%A7%C3%A3o_Karina_CIN.pdf>. Acesso em 23 fev. 2019.

COUTINHO, Paulo César. **REST Tutorial**. 2013. Disponível em: <https://www.devmedia.com.br/rest-tutorial/28912>. Acesso em 10 abr. 2019.

DATAGRIP. **DataGrip**. Disponível em <https://www.jetbrains.com/datagrip/>. Acesso em 06 nov. 2019.

DATE, C J. **Introdução a sistemas de banco de dados**. 8º edição. Rio de Janeiro, RJ. Elsevier. 2003.

DBEAVER. **DBeaver, universal database client**. Disponível em <https://dbeaver.com/>. Acesso em 02 nov. 2019.

DOCKER, **Docker**. Disponível em <https://www.docker.com/>. Acesso em 03 nov. 2019.

DUARTE, Felippe Roberto Bayestorff. **Integração de Sistemas**. 2009. Disponível em: <https://repositorio.ufsc.br/handle/123456789/184496>. Acesso em 30 abr. 2019

ELMASRI, Ramez; NAVATHE Shamkant B. **Sistemas de banco de dados**. Person Education do Brasil, 7ª edição. 2018.

HASURA, **Hasura**. Disponível em <https://hasura.io/>. Acesso em 06 nov. 2019.

FONSECA, J. J. S. **Metodologia da pesquisa científica**. Fortaleza: UEC, 2002.

FUSIO, **Fusio**. Disponível em <https://www.fusio-project.org/>. Acesso em 03 nov. 2019.

GOMES, Daniel Adorno; JANDL, Peter Jr. **WEB SERVICES SOAP E REST**. 2009. Intellectus, Ano V, Nº6.

HEUSER, Carlos Alberto. **Projeto de banco de dados**. Bookman, 6º Edição. 2009 282p.

JAVASCRIPT. **Documentação**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em: 22 de abr. 2019.

KUEHNE, Bruno Tardiole. **Modelos e algoritmos para composição de Web Services com qualidade de serviço**. 2009. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-05042010-111224/publico/bruno.pdf >. Acesso em 26 fev. 2019.

MARCONI, Marina de Andrade; LAKATOS, Eva Maria. **Fundamentos de Metodologia Científica**. 5ª Edição. São Paulo: Editora Atlas, 2003.

MARTINS, Victor Manuel Moreira. **Integração de Sistemas de Informação**, 2005. Disponível em : <https://repositorium.sdum.uminho.pt/bitstream/1822/5657/3/tese_mestrado_victor_martins_2005.pdf>. Acesso em 24 abr. 2019.

MATERIALIZE. **Documentação**. Disponível em: <https://materializecss.com/>. Acesso em 22 abr. 2019.

MDN, Mozilla Developer Network. **Array**. 2017. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array#DescriC3%A7%C3%A3o>. Acesso em 26 fev. 2019.

MITCHELL, Lorna Jane. **PHP Web Services**. O'RELLY, 1ª edição. 2013.

MONGO, Documentação MongoDB. **Manual**. Disponível em: <https://docs.mongodb.com/manual>. Acesso em 22 abr. 2019.

PALMEIRA, Thiago Vinícius Varallo. **Introdução à Programação Orientada a Objetos em Java**. DevMedia. 2012. Disponível em <https://www.devmedia.com.br/introducao-a-programacao-orientada-a-objetos-em-java/26452>. Acesso em 09 nov. 2019.

PANIZ, David. **NoSQL: Como armazenar os dados de uma aplicação moderna**. Casa do Código. São Paulo, SP. 2016.

PGADMIN. **pgAdmin**. Disponível em <https://www.pgadmin.org/>. Acesso em 02 nov. 2019.

PHP. **O que é o PHP**. Disponível em: <https://secure.php.net/manual/pt_BR/intro-whatis.php> . Acesso em 25 fev. 2019.

PHPMYADMIN. **phpMyAdmin**. Disponível em: <https://www.phpmyadmin.net/>. Acesso em 02 nov. 2019.

POSTGRESQL. **O que é o PostgreSQL?**. Disponível em: <https://www.postgresql.org/>. Acesso em 20 abr. 2019.

SADALAGE, Pramod J.; FOWLER, Martin. **NoSQL essencial: um guia conciso para o mundo emergente da persistência poliglota**. São Paulo: Novatec, 2013.

SANTOS, Antonio Raimundo dos. **Metodologia científica: a construção do conhecimento**. 2ª ed. Rio de Janeiro: DP&A editora, 1999.

SAUDETE, Alexandre. **REST: Construa API's inteligentes de maneira simples**. 2014. Casa do Código. eBook Kindle.

SCHREIBER A. Th; SCHREIBER, Guus; AKKERMANS, Hans; ANJEWIERDEN, Anjo; SHADBOLT, Nigel; HOOG, Robert; VELDE, Walter Van; WIELINGA, Bob. **Knowledge engineering and management: the CommonKADS methodology**. Cambridge/ Massachussets: MIT Press, 2002. 932 p.

SILBERSCHATZ, Abraham; KORTH; Henry F.; SUDARSHAN; S. **Sistema de banco de dados**. 6º edição. Rio de Janeiro, RJ. Elsevier. 2012.

SILBERSCHATZ, Abraham; KORTH; Henry F.; SUDARSHAN; S. **Sistema de banco de dados**. São Paulo, SP. MAKRON Books. 1999.

SOMMERVILLE, Ian. **Engenharia de software**. 9. ed. São Paulo, SP: Pearson Prentice Hall, 2011. 529 p.

VICCI, Claudia. **Banco de dados**. 1º Edição. São Paulo, SP. Person Education do Brasil. 2014.

WAINER, Jacques. **Métodos de pesquisa quantitativa e qualitativa para a Ciência da Computação**. In: KOWALTOWSKI, Tomasz; BREITMAN, Karin; organizadores. Atualizações em Informática 2007. Rio de Janeiro: Ed. PUC-Rio; Porto Alegre: Sociedade Brasileira de Computação, 2007.

WELLINGTON, Balbo De Camargo. **Interfaces: Programação Orientada a Objetos**. DevMedia. 2010. Disponível em <https://www.devmedia.com.br/interfaces-programacao-orientada-a-objetos/18695>. Acesso em 09 nov. 2019.

APÊNDICES

APÊNDICE A – Questionário de avaliação da ferramenta desenvolvida

01. Você trabalha com desenvolvimento de software?

( ) Sim
( ) Não
( ) Indiretamente

02. Quantos anos de experiência você possui na sua profissão?

( ) 1 ano ou menos
( ) 1 a 2 anos
( ) 2 a 4 anos
( ) 4 a 6 anos
( ) 6 anos ou mais

03. Para você a usabilidade do software é ?

( ) Totalmente apropriada
( ) Parcialmente apropriada
( ) Regular
( ) Parcialmente inapropriada
( ) Totalmente inapropriada

04. As nomenclaturas utilizadas na ferramenta, considerando seu entendimento, estão ...

( ) Totalmente adequada
( ) Parcialmente adequada
( ) Regular
( ) Parcialmente inadequada
( ) Totalmente inadequada

05. Você usaria esta ferramenta nos seus projetos ?

( ) Sim, apenas com as funcionalidades existentes
( ) Sim, com mais funcionalidade
( ) Não, pois as funcionalidades não me ajudariam
( ) Não, pois faltam funcionalidades importantes

06. Quanto às funcionalidades existentes na ferramenta elas são ...

( ) Suficientes, atendem a proposta
( ) Suficientes, mas com ressalvas, pois faltam alguns recursos
( ) Insuficientes, ferramenta incompleta

07. Ter uma API com esses recursos, contribui para o desenvolvimento de software ?

( ) Sim
( ) Não

08. Você percebe que existe a dificuldade em trabalhar com os diferentes tipos bancos de dados e uma ferramenta como essa pode ajudar nesse processo?

( ) Sim
( ) Não

09. A documentação da API ficou clara? Conseguiu identificar os recursos que a API fornece para buscar registros?

( ) Sim
( ) Não

10. Destaque pontos positivos na ferramenta.

11. Destaque pontos negativos na ferramenta.

12. Registre aqui qualquer comentário, crítica ou sugestão sobre a ferramenta.

APÊNDICE B – Roteiro para utilização da ferramenta desenvolvida

1. Acesse a ferramenta pelo link: http://45.77.210.153/mybases/
    - login: admin
    - senha: admin

2. Crie um conector do tipo PostgreSQL, utilizando os dados abaixo**. Verifique se a conexão funcionou**.
    - descrição: **postgres**
    - endereço: 206.189.235.54
    - nome: dbteste
    - usuário: teste
    - senha: teste123

3. Crie um conector do tipo Mongo, utilizando os dados abaixo**. Verifique se a conexão funcionou**.
    - descrição: **mongo**
    - endereço: 206.189.235.71
    - nome: teste_mongo
    - usuário: teste
    - senha: teste123
    - porta: **27017**

4. Crie um consumidor chamado **teste-api**.
    - defina uma key de acesso **testetcclucas**
    - Libere permissões de leitura e escrita para ambos conectores.

5. Utilize o Teste da API para **buscar** os registros do conector **postgres**, da estrutura de pessoas que contenham a palavra “Lucas” no atributo nome, ordenando por esse atributo em **ordem ascendente**, Para isso você pode utilizar o trecho de código em PHP disponível na ferramenta. Verifique a documentação para identificar como é realizado uma ordenação.

6. Efetue a inserção do seu nome através do atributo nome no conector **mongo** da estrutura de pessoas. Você pode utilizar qualquer uma das linguagens disponíveis para Teste da API.

7. Utilize o Teste da API para buscar os registros do conector **mongo**, criado anteriormente, da estrutura de pessoas, **limitando** 3 registros, Para isso você pode utilizar o trecho de código em NodeJS disponível na ferramenta. Verifique a documentação para identificar a forma de estabelecer limites.

8. Conceda a permissão de escrita, leitura e manipulação, para o grupo ADMIN no módulo administrativo, para ambos conectores criados.

9. Com o auxílio do **explorador de base de dados**, faça uma busca pelos registros da base **mongo**, identifique o **_id** que contenha o seu nome.

10. Com o auxílio do **Teste da API** efetue a **exclusão** deste registro. Utilize a linguagem PHP fornecida pela ferramenta e se necessário utilize a documentação.

11. Com o auxílio de teste da API, efetue a **busca** de uma pessoa, cujo o código identificador é **1** no conector **postgres**, na estrutura pessoas, copiando e colando o código gerado pela ferramenta na linguagem Shell, no seu terminal.

12. Crie uma estrutura utilizando o **explorador de base de dados** no conector **postgres**, com o nome **teste_tcc**, somente com a coluna ID que é uma chave primária do tipo inteiro e depois exclua a mesma.

13. Gere um .csv com as requisições **todas as requisições** feitas **hoje**.